

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION

AD-A218 411

Approved
Do. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		3. DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	
2a. SECURITY CLASSIFICATION AUTHORITY		5. MONITORING ORGANIZATION REPORT NUMBER(S) AFIT/CI/CIA-89-135D	
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE		7a. NAME OF MONITORING ORGANIZATION AFIT/CIA	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6583	
6a. NAME OF PERFORMING ORGANIZATION AFIT STUDENT AT ARIZONA STATE UNIVERSITY	6b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
6c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS	
8a. NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	PROGRAM ELEMENT NO.	PROJECT NO.
8c. ADDRESS (City, State, and ZIP Code)		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) (UNCLASSIFIED) Supporting Selection Decisions Based on the Technical Evaluation of Ada Environments and Their Components			
12. PERSONAL AUTHOR(S) Patricia K. Lawlis			
3a. TYPE OF REPORT DISSERTATION	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1989	15. PAGE COUNT 472
16. SUPPLEMENTARY NOTATION APPROVED FOR PUBLIC RELEASE IAW AFR 190-1 ERNEST A. HAYGOOD, 1st Lt, USAF Executive Officer, Civilian Institution Programs			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
DTIC ELECTE FEB 01 1990 S B D			
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL ERNEST A. HAYGOOD, 1st Lt, USAF		22b. TELEPHONE (Include Area Code) 513-255-2259	22c. OFFICE SYMBOL AFIT/CI

**SUPPORTING SELECTION DECISIONS
BASED ON THE TECHNICAL EVALUATION OF
ADA ENVIRONMENTS AND THEIR COMPONENTS**

by

Patricia K. Lawlis

**A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy**

ARIZONA STATE UNIVERSITY

December 1989

Key References

Department of Defense, *E&V Reference Manual*, Version 1.1, October 1988.

Department of Defense, *Requirements for Evaluation and Validation of Ada Programming Support Environments*, Version 2.0, November 1986.

J. Foreman and J. Goodenough, *Ada Adoption Handbook: A Program Manager's Guide*, Version 1.0, CMU/SEI-87-TR-9, Pittsburgh: Software Engineering Institute, May 1987.

S. French, *Decision Theory*, New York: John Wiley & Sons, 1986.

R. C. Houghton and D. R. Wallace, "Characteristics and functions of software engineering environments," *Software Engineering Notes*, vol. 12, no. 1, January 1987.

T. G. L. Lyons and J. C. D. Nissen, *Selecting an Ada Environment*, New York: Cambridge University Press, 1986.

J. McDermid and K. Ripken, *Life Cycle Support in the Ada Environment*, New York: Cambridge University Press, 1984.

M. H. Penedo and W. E. Riddle, "Guest editors' introduction: software engineering environment architectures," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.

R. S. Pressman, *Software Engineering*, Second edition, New York: McGraw-Hill, 1987.

W. E. Souder, *Management Decision Methods*, New York: Van Nostrand Reinhold, 1980.

N. Weideman, *Ada Adoption Handbook: Compiler Evaluation and Selection*, Version 1.0, AD A207717, Software Engineering Institute, March 1989.

M. Zeleny, *Multiple Criteria Decision Making*, New York: McGraw-Hill, 1982.

on For	
RA&I	☒
3	☐
iced	☐
ation	



Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

SUPPORTING SELECTION DECISIONS
BASED ON THE TECHNICAL EVALUATION OF
ADA ENVIRONMENTS AND THEIR COMPONENTS

by

Patricia K. Lawlis

has been approved

August 1989

APPROVED:

Timothy E. Lindquist , Chairperson
Robert H. Fairer
Richard J. [unclear]
Paul C. [unclear]
James S. Callagher
Supervisor Committee

ACCEPTED:

Robert E. Barak
Department Chairperson

Brian L. Foster
Dean, Graduate College

Abstract

At present, most software selection decisions are based on data which cover only a very small portion of the scope of software assessments, and this amount of data is inadequate for making intelligent decisions. Only a computer system such as the one designed in this research can bring together an adequate amount of assessment data with a broad enough scope and provide a decision maker with a consistent and comprehensive analysis of such data so that intelligent decisions are possible.

The research described in this dissertation has been a start, but only a start, toward developing a comprehensive Ada Programming Support Environment (APSE) assessment capability. The current state of assessment technology has been examined, and it has provided the basis for the evaluation framework designed. This framework is an important and necessary step in determining the type of assessment data which must be provided in order to make a comprehensive software evaluation. Designing a prototype computer system for APSE selection has demonstrated the viability of using this framework to improve the software selection process. A variation of the design could also be used when selecting any type of software in general.

The Ada Software Selection assISTant (ASSIST), the product of this research, has the potential for becoming a very important part of software evaluation technology. In addition to providing a design and a prototype for a capability to assist a decision maker in

selecting software, it also provides an initial organizational framework on which expectations for future assessments can be based.

In loving memory of my mother,
whose unconquerable spirit has always been my inspiration.

Acknowledgements

First and foremost, thanks to my committee, Jim Collofello, Bob Fainter, Paul Jorgensen, and Dick Peschke, who have provided just the right mixture of assistance and thought-provoking ideas. And a special thanks to my advisor, Tim Lindquist, who talked me into attending Arizona State University, and then provided sound advice from beginning to end.

Thanks also goes to the entire E&V Team for their support and sponsorship, with special recognition to Ray Szymanski, Becky Abraham, Mike Burlakoff, Bard Crawford, Jay Ferguson, Greg Gicca, Marlene Hazle, Tom Leavitt, Ronnie Martin, John McBride, Gary McKee, Sandi Mulholland, Helen Romanowsky, Lloyd Stiles, and Nelson Weiderman.

Thanks to the Air Force, and especially to David Lee, my former boss at the Air Force Institute of Technology (AFIT), for the encouragement and for making this a tour of duty. Thanks to my family for understanding. For special support, thanks to David Struble of Texas Instruments and James Holt of AFIT. For special friends, thanks to Kathy Austin, Rick Gross, and Debbie Olds. For service above and beyond, thanks to my good friend and colleague Karyl Adams, who was always there.

Table of Contents

	Page
List of Figures	ix
List of Acronyms	x
 1. Problem	 1
Introduction.....	2
Thesis.....	5
Scope of the Research.....	7
Approach.....	7
Overview.....	12
 2. Background	 13
Software Environments	13
The Ada Programming Support Environment (APSE) ..	16
Software Evaluations	19
The Decision Support System (DSS).....	21
System Design.....	25
 3. System Requirements.....	 31
Introduction.....	31
Questionnaire	36
The Requirements Document	37
The Prototype	40
Reviewing the Prototype.....	50
Back to the Requirements Document.....	51
 4. System Design.....	 54
Object-Oriented Design.....	55
HyperCard Support and Drawbacks	62
Developing a Common Terminology.....	67
Getting Input from the Decision Maker.....	73
Providing Recommendations to the Decision Maker....	93

	Page
5. Evaluation.....	106
Evaluating the ASSIST Design.....	107
A Case Study of the ASSIST Prototype.....	129
Final Analysis.....	165
6. Conclusions and Recommendations.....	167
Accomplishments.....	167
Future Directions.....	169
Summary.....	174
References	176
Appendix	
A Software Evaluation Questionnaire.....	193
B Notes on the ASSIST Prototype Version 1.0.....	196
C Requirements for ASSIST.....	198
D Test Plan for ASSIST.....	217
E Test Descriptions.....	231
F Requirements Cross-Reference Matrix.....	266
G Design Documentation for ASSIST.....	289
H Features Structure for ASSIST Prototype Version 2.0.....	324
I Criteria Structure for ASSIST Prototype Version 2.0.....	342
J Glossary.....	358
K Suggested Features and Criteria by Application.....	376
L User Manual for ASSIST Prototype Version 2.0.....	383
M Windows Seen in Management-Oriented Scenario.....	429
N Windows Seen in Technically-Oriented Scenario.....	444
O Notes on the ASSIST Prototype Version 2.0.....	458

List of Figures

Figure		Page
2.1	Layers of support in a software environment.....	15
2.2	The components of an APSE	18
2.3	The components of an expert system.....	22
2.4	The components of a DSS	23
2.5	Waterfall model of software development.....	26
3.1	The structure of the ASSIST subsystems.....	38
3.2	Relationships among ASSIST activities.....	49
4.1	Object-oriented design components.....	56
4.2	ASSIST Design Documentation card.....	57
4.3	ASSIST Program Design Language (PDL) card.....	59
4.4	ASSIST Design Diagram card	60
4.5	ASSIST Interfaces card.....	61
4.6	Top level structure of characteristics for ASSIST prototype	71
4.7	Icons used for the support function buttons.....	78
4.8	The graphical browser	81
4.9	Algorithm for calculating an overall feature rating.....	97
4.10	Calculations for an overall feature rating.....	98
4.11	Algorithm for calculating an overall rating for one product.....	101
5.1	Important generic data to be collected on features.....	132
5.2	Important data to be collected on features specific to compilation systems.....	133
5.3	Important data to be collected on criteria.....	134
5.4	Breakdown of scenario actions.....	163

List of Acronyms

ACEC	Ada Compiler Evaluation Capability
ACM	Association for Computing Machinery
AES	Ada Evaluation System
AJPO	Ada Joint Program Office
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ASSIST	Ada Software Selection assISAnt
CAIS	Common APSE Interface Set
CASE	Computer aided software engineering
DBMS	Database management system
DGMS	Dialog generation and management system
DoD	Department of Defense
DSS	Decision support system
E&V	Evaluation and validation
IEEE	Institute of Electrical and Electronics Engineers
IPSE	Integrated project support environment
KAPSE	Kernel APSE
MAPSE	Minimal APSE
MBMS	Model-based management system
MIS	Management information system
MOD	Ministry of Defence
PCTE	Portable Common Tool Environment
PDL	Program design language
PIWG	Performance Issues Working Group

PSE	Programming support environment
SEE	Software engineering environment
SIGAda	Special Interest Group on Ada

1. Problem

Software development technology has reached a state where there is a proliferation of software for almost any purpose. If it does not yet exist, but somebody wants it, it will soon appear, in some form or another. However, technology has a long way to go before it can provide an effective method for evaluating all this software. Vendors' descriptions purposely use different terminology and avoid common ground, thus making comparisons of competing products very subjective. Trying to assess whether a piece of software meets a specific set of user requirements is often extremely difficult because no uniformly applicable method for software evaluation exists.

The Department of Defense (DoD) needs the ability to evaluate new Ada products as they become available, and so it is trying to advance the state of software evaluation technology. Particular emphasis is on the software which will comprise the Ada Programming Support Environment (APSE). The DoD-sponsored APSE Evaluation and Validation (E&V) Team has laid much of the groundwork for advancing evaluation technology, but much work remains to be done before this technology is usable.

This research helps advance software evaluation technology to a useful state by developing a mechanism for applying uniform criteria to the software selection process. A software system which embodies this mechanism has been designed to assist the technical manager in making Ada software selection decisions. This document

details the design and prototyping of the software system, called the Ada Software Selection assISAnt (ASSIST)

Introduction

Current research is laying the foundation for understanding the evaluation technology needed, but no work has yet developed a complete framework from which software evaluation decisions can be made. Nidiffer [114] has studied this decision process from a management perspective of making investment strategy decisions. However, the decision process he has developed does not adequately address the technical aspects of the problem. He expects simply to be able to classify software environments by "type," making the underlying assumption that all environments of a particular type can meet specific requirements. Such an assumption can prove disastrous when the technical capabilities of the environments of any one type can differ significantly, and a major investment may be made to purchase an environment which cannot support all aspects of a project.

The DoD's APSE E&V team has produced evaluation requirements and a taxonomy of evaluation criteria to be used in assessing the capabilities of an environment and its component software [50], [52]. From this a guidebook has been developed with pointers to existing APSE evaluators (test suites, checklists, etc.) [49]. Among these evaluators is the suite of tests developed by the E&V team for Ada compiler performance evaluation [44]. Other organizations, such as the Performance Issues Working Group (PIWG)

of the Special Interest Group on Ada (SIGAda) of the Association for Computing Machinery (ACM), have also developed test suites [92], [113], [129], [150]. Most of these also test compiler performance evaluation. The Ministry of Defence (MOD) in the United Kingdom has sponsored the development of a system called the Ada Evaluation System (AES). Its current version covers the evaluation of Ada compilers, linkers, loaders, symbolic debuggers, and program library management, and later versions are expected to cover more APSE tools [123], [124], [162]. However, there is no coordination among all these evaluation vehicles, and their scope is extremely limited.

The software evaluation process is very complex. Once evaluation information is available from various assessors (evaluators), assessment has just begun. To be meaningful, an evaluation must be relevant to the application of interest, and it must be comprehensive. An evaluation addressing only one aspect of the software is not complete and could be misleading. Various attributes of the software determine whether it is good for a particular situation or application. Because a program which is "good" in one situation is not necessarily adequate under different circumstances, a piece of software cannot be evaluated properly by just awarding it a simple rating. To say that compiler A is a 7 on a scale from 1 to 10, without putting it in a proper context, is meaningless. Likewise, to say editor B is better than editor C means nothing unless it is qualified, as in "with respect to attribute X."

If sound software purchasing decisions cannot be made based on evaluation information, that information is of little value to the technical manager. The information available from current evaluation test suites is very difficult to use in the decision making process. It is complex and typically voluminous. All results from each of the numerous steps required to execute the test suite are reported. Summaries, when provided, usually still require the expertise of a statistician to understand. Furthermore, they provide only a very small piece of the evaluation picture, with no perspective on the rest of the picture, and often with no indication that further assessment is desirable [44], [123], [124].

In addition to performance statistics, text-like evaluation information is sometimes available as checklists, descriptive narrative, and the like. This makes an evaluation more comprehensive, but it also adds to the amount of information to be digested by the decision maker. To gain a realistic and useful picture from so much information, the time investment required would be prohibitive, using currently available evaluation technology. Furthermore, even if the necessary time were invested, it is virtually impossible for a person to keep track of the pertinent "big picture" (ignoring the details), while applying selection criteria consistently.

The only practical way to achieve intelligent decisions based on comprehensive, consistent, concise, and timely evaluation information is to let a computer do the menial work. However, although the computer is very good at providing information, it does

not automatically provide meaningful information for the human decision maker. This task still requires much effort.

Thesis

An interactive computer system of the type usually called a decision support system (DSS) is a viable approach to giving the decision maker the ability to make both timely and informed Ada software selection decisions. The purpose of this research was to show that such a system can appropriately organize and summarize all of the available evaluation information on APSEs and present it to a decision maker in an effective and timely manner.

Since software evaluation technology is so new, there is no broad base of knowledge from which to work in developing a DSS for software evaluations. Research was required to bring together the work in this area and blend it into a comprehensive framework supporting the presentation of evaluation information. Such research took the form of the design and prototyping of a DSS, called ASSIST, which contains the necessary framework. It investigated various forms of evaluation information which could be put into the DSS database, the types of analyses which could be performed on such data, the pertinent views of information which could be derived from the DSS, and the effective organization and forms of representations for those views. The design incorporated a framework for handling evaluation information of all types and for organizing understandable and useful presentations of such information.

To demonstrate the DSS concept, the research concentrated on three main areas. The first was developing a process for getting information from the decision maker on the characteristics of the software to be selected and the evaluation criteria to be used in the selection process. The second was developing a method for organizing and summarizing the recommendations and other information ASSIST would provide to the user. Finally, ASSIST used the assessment of Ada compilation systems as its main research subject. Although this is only a small part of the possible types of Ada software which could be selected using ASSIST, this area was chosen because it is the one in which most current evaluation efforts have concentrated. The development of this assessment process made the research timely, and it also resulted in the examination of evaluation information provided by actual test suites.

The research concluded with an evaluation of the resulting system design. This was accomplished through analysis and the use of scenarios which were run on the prototype. The analysis evaluates how well the design supports the decision maker's specification of the software to be selected, how well it works with various forms of evaluation information in its database, and how well it organizes and summarizes the recommendations and other output in support of the decision maker. The scenarios follow example sessions run on the prototype, demonstrating an implementation of the design and the usability of the resulting DSS.

Scope of the Research

Several products were developed as a part of this research. All design information produced is a part of the system documentation. This includes all products from requirements through prototype implementation.

The dissertation document includes a description of all aspects of the research, including an evaluation of the system design. This evaluation comprises an analysis of the design with respect to all aspects of APSE evaluation as well as the description of the scenarios run on the prototype.

Prototype software is another product of this research. The prototype is a working model, but it is set up to provide only limited evaluation information on Ada compilation systems.

In the process of designing ASSIST, it was clear that much of the design is applicable to software evaluations in general. Care was taken to design reusable components which could make up a significant part of a more general software evaluation DSS. However, the focus of the research was on the spectrum of APSE evaluations to be considered.

Approach

The research concentrated on the appropriate specification of the software evaluation parameters to be used, the analysis of the evaluation information in the system database, and the presentation form of the recommendations and other information to the decision maker. Efficiency issues were not a major concern in developing the

prototype, since this was for proof of concept only. Three distinct steps to this research were:

- 1) developing the requirements for the ASSIST software,
- 2) systematically incorporating the various aspects of current evaluation technology into the design of ASSIST, and
- 3) evaluating the resulting design.

Requirements

The first step toward designing a computer system is to determine its requirements. That was the first major effort in this research. A set of requirements had to include a detailed list of the functions the system had to perform. In addition, it also had to include *performance characteristics*, design constraints, and system attributes.

It is most difficult to specify requirements for a system which is very different from any which has been previously implemented. Although ASSIST was identified early on as a type of DSS, it is still different from the "normal" DSS because it attempts to combine both technical and management issues. The general structure of a DSS was determined to be appropriate for the system, so the major subsystem structure was decided early. However, developing more detailed requirements proved to be much more difficult.

Prototyping a system has proven to be a very effective technique for help in understanding a system well enough to determine its detailed requirements [7]. The ASSIST prototype was started very early for this reason, and it was valuable in providing

the feedback to permit the development of appropriate detailed system requirements. The system used to develop the prototype was a Macintosh II™ running HyperCard™. The rich user interface resources available in HyperCard made this a very attractive choice, and it proved to be a good one. It permitted the creation of good user interfaces while expending very little time and effort. This allowed the research to focus on the more technical aspects of the DSS.

Design

Once the requirements were completed in detail, it was time to focus on the system design. This was really the most significant aspect of the research effort. The main objective was to bring various aspects of current knowledge about software evaluation together in a software system. The system would use evaluation information stored in its database to make comparisons of different implementations of the same type of software. Comparisons would use appropriate methods for the different types of evaluation information, and they would focus on user specified characteristics of importance. This system would then provide advice on Ada software selections to a technical manager.

Object-oriented design and hypertext were determined to be appropriate techniques for use in the system design. These fit well with the use of HyperCard as the prototyping tool, since HyperCard implements a type of hypertext, and the HyperCard language, HyperTalk™, is object-oriented. The system was designed according

to the basic structure of a DSS. The independent subsystems approach permitted the design to concentrate on the decision logic subsystem and the structure of the knowledge base, which were the areas of real significance in this research. The knowledge base subsystem was developed only as much as necessary to design the knowledge base structure. The user interface subsystem was not designed and developed. Likewise, an acquisition facility for putting information into the database and knowledge base was also not designed and developed. In the prototype, the existing HyperCard software was used for these subsystems, and the effort was then able to concentrate on developing appropriate interfaces and knowledge representations.

Evaluation

It is important to be able to evaluate the products of this research to determine how much has been accomplished. Such an evaluation also provides a useful summary of the research results. This assists researchers who may continue in this area and helps to avoid duplication of effort.

The most effective evaluation of a DSS design is via the use of the implemented system [4], [133]. If a product is developed for commercial use, it will go through many iterations from prototype through the product stage, and it can benefit from continual feedback from users. This feedback provides the evaluation of how useful and complete the DSS is. Anything that is missing will be noted, areas which need improvement will be identified, suggestions will be

made, and features which are useless will also be identified. If the entire system is useless, it will be immediately apparent that it is not being used in the initial prototype stage, and it can either be discarded or completely redone.

For a design which is the product of an individual research effort, this type of feedback was not possible. Some selected users were able to provide feedback on the design as it progressed, but this was very limited, and it was not the type of information which was conducive to assessing the accomplishments of the research in any complete sense. Hence, a different approach was required.

The two types of assessment which make the most sense for this type of research are (1) provide a technical analysis of the system design and (2) do a case study using the prototype to show how a part of the design implementation works. The analysis of the design must look at what the system should be able to accomplish and the technical aspects of APSE evaluations which should be included in the system. It must then determine whether all technical aspects have been considered, as well as whether the system can accomplish its objectives. Another part of the design analysis should be how well the system is structured to permit modifications and enhancements as the technology matures.

The case study approach was to run scenarios on the working prototype, covering a different aspect of design evaluation. This cannot really determine if the design is as complete as it should be, since not all capabilities included in the design will be used in the case study. In fact, many will not even be implemented in the

prototype. However, a case study illustrates just how usable a system is in a way that cannot really be determined just by analyzing the design. If the scenarios reveal that running the DSS is awkward or ineffective, then either the design is at fault or else the user interface has not been implemented well enough to be able to make a useful assessment of the design. In the case of this research, the user interface was provided by a viable commercial product, so it was possible to make a good assessment of the usability attributes of the system design.

Since the two evaluation approaches produce different types of evaluation information, both approaches were used in assessing this research. This provided the best possible assessment of how well the research had accomplished its goals.

Overview

The next chapter provides background information concerning all of the various areas of research which had to be investigated in this research effort. Chapters 3 and 4 detail the requirements and design of ASSIST, with each chapter including descriptions of pertinent portions of the development of the ASSIST prototype. Chapter 5 provides the evaluation of this research effort. Finally, Chapter 6 provides conclusions and recommendations derived from the research.

Note: Macintosh, HyperCard, and HyperTalk are trademarks of Apple Computer, Inc.

2. Background

This research required an in-depth understanding of several areas. Since the APSE is a software environment for Ada, the general area of software environments was examined at length, along with how the APSE compares with other environments. Evaluation technology was another important area for examination, particularly recent efforts toward developing the specific technology for evaluating APSEs and their components. The research involved designing a decision support system, so it was also important to examine the principles and techniques which have been developed for DSS. These must also be compared and contrasted with other types of systems to understand why the DSS was appropriate for this research. Finally, appropriate design techniques used in developing the DSS were examined. The following sections summarize the important ideas drawn from these areas.

Software Environments

After programmers began building tools to aid them in their work, the next step was for a collection of tools to be brought together to provide a programming "environment". The idea of most software environments today is that the whole is greater than the sum of the parts. Put succinctly, programming environments are "computer-aided design systems for software" [16].

Environments can be general purpose or application specific. Existing environments range from those which are programming-oriented operating system extensions to complex sets of tools which

cover all life-cycle activities. Many different names are also typically given to environments, particularly those intended to support large-scale software development. These include programming support environment (PSE), software engineering environment (SEE), computer-aided software engineering (CASE) environment, and integrated project support environment (IPSE). Some interpret various shades of meaning in these names, but in use they are essentially synonymous.

It is useful to think of a software environment in terms of layers of support. A very complete description of this layering is given by Penedo and Riddle [120] (see Figure 2.1). At the lowest layer is the hardware, usually with a native operating system running on top of it. This is followed by a layer of environmental support, usually consisting of a virtual operating system and systems for object management, user interface management, and environment management. Next is a tool/capability layer providing the environment functionality, often including an integration mechanism. On top of these layers are the tools of the environment. These tools can be of many types, and they may be in layers as well. They can be generally classified as adaptation support tools (used to define and generate project-specific environments) and project user support tools. The most important thing is that they support the various activities of a specific environment.

Activities supported within a software environment can be both technical and managerial in nature. While many programmers tend to see an environment as comprising only those tools necessary

for the actual process of code generation (compiler, editor, linker, loader, etc.), those interested in engineering large software systems generally recognize that an environment should provide as much support as possible for all functions involved in the software system development process. This includes the user interface and environment database as well as managerial functions, such as configuration management and quality assurance, and technical functions, ranging from system concept through system retirement [34], [36], [55], [68], [90], [101], [102], [107], [125], [146], [153].

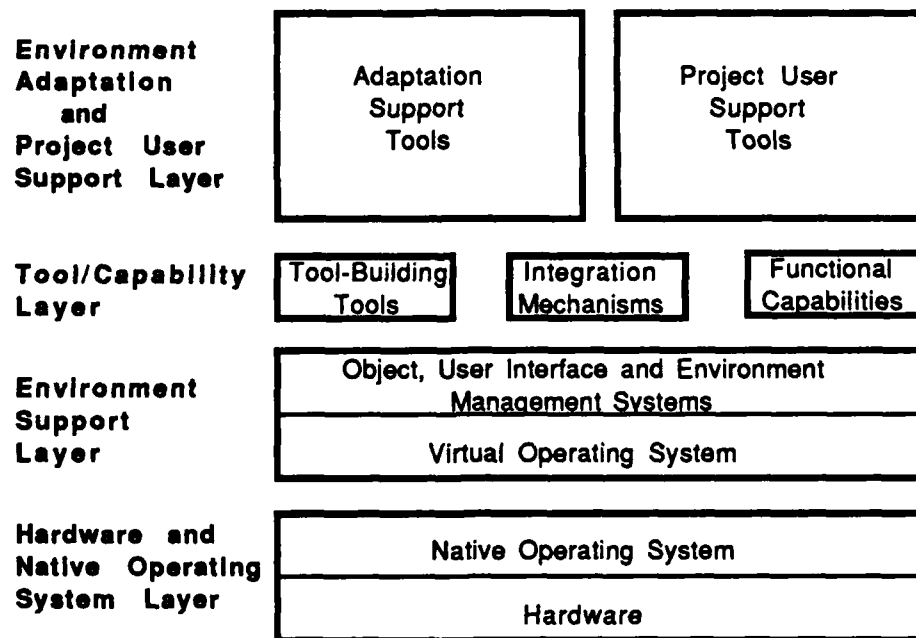


Figure 2.1 - Layers of support in a software environment [120]

The integration of the components of the environment is also an important issue, and various levels of integration can be defined. At one extreme, the integration can be very loose, where the tools share a common operating system, but there is no other guarantee

that the tools can work together, share common data, etc. At the other extreme, the tools interact with the native system (hardware and, usually, a native operating system) via a well-defined interface. In this case, the environment maintains a database for all the external information manipulated by the tools. This permits all data to be kept in a common format and, thus, tools can interact directly with one another. On a larger scale, this also permits transportability (tools can be shared) and interoperability (database information can be shared) among different environments, independent of their native system configurations [70], [76], [82], [88], [109], [116], [130], [142], [148], [156], [158], [167], [168].

It is this concept of a highly integrated environment which was originally defined for the APSE [29]. However, those environments labelled APSEs today range from very loosely integrated to moderately integrated. The only true identifying characteristic of current APSEs is the inclusion of a validated Ada compiler.

The Ada Programming Support Environment (APSE)

During the development of the Ada language, it was recognized that just another language was not going to solve the "software crisis," no matter how well designed that language might be. So the DoD defined an environment to go with Ada. It was called the Ada Programming Support Environment, or APSE, and its requirements were laid out in a document usually referred to as STONEMAN [29]. This document gives general guidance for developing an APSE which

will support, among other things, transportability and interoperability among its various implementations.

Theoretically, an APSE is a highly integrated environment. It has three major components -- the tools, the interfaces, and the database. Figure 2.2 illustrates the APSE structure, which is usually depicted as layers of an "onionskin" rather than in the form of Figure 2.1. However, this is a comparable, though not so general, structure. All system and implementation dependencies are kept in a kernel, called the Kernel APSE or KAPSE. Outside the KAPSE is the minimal set of tools which can be put together in an APSE to meet the STONEMAN requirements. An APSE containing just this minimal set of tools interacting with the KAPSE is referred to as the Minimal APSE or MAPSE. The MAPSE uses the services of the KAPSE only through a standard, system independent interface. STONEMAN does not attempt to define the standard interface, but other efforts have worked on that, resulting in the definitions of the Common APSE Interface Set (CAIS) and the Portable Common Tool Environment (PCTE) [29], [37], [46], [65], [95], [117].

The concept for a mature APSE is to have many tools added to the minimal tools of the MAPSE. As more and more sophisticated tools are added and integrated with other tools, the power of the APSE continues to increase. For example, management support tools can provide automated support for all aspects of managing a project developed on the APSE, resulting in better informed and more timely management decisions. Likewise, powerful development tools can

improve productivity tremendously [62], [63], [69], [96], [104], [108], [145], [157].

While it is desirable for an APSE to be fully integrated and provide the user with much power, in reality, no APSEs currently exist which are really mature enough to provide much more than code generation support. Hence, it is practical to consider even a small tool set, consisting of an Ada compiler and those tools used with it to produce code, as an APSE for the purposes of this research. However, that is not to say that the more powerful APSE concept was abandoned. For completeness it was important for this research to address all possible APSE structures, from the least to the most powerful.

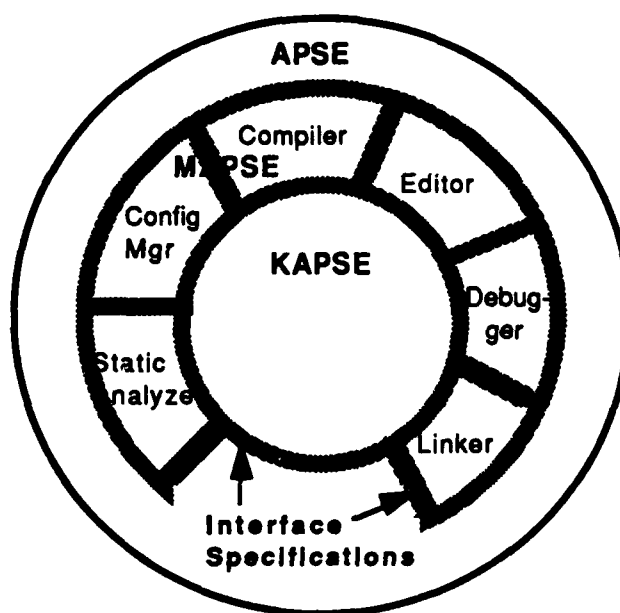


Figure 2.2 - The components of an APSE [29]

One of the aspects of particular importance when characterizing an APSE is knowing which of several standards it may support. Standards are defined and used for various reasons. Some of those of interest in an APSE are software development standards and interface standards of various types, including APSE interfaces such as the CAIS and the PCTE as well as more generally applicable ones such as graphics standards.

Software Evaluations

Soon after the Ada Joint Program Office (AJPO) was formed to oversee the introduction of Ada into DoD software programs, the AJPO established several tasks to be accomplished through volunteer teams. One of these was the APSE Evaluation and Validation Task, whose purpose is to further the state of evaluation and validation technology. The E&V Team, established to address the task, brought together interested government personnel, and they were joined by numerous "distinguished reviewers" from industry and academia.

This unique body of expertise has provided valuable contributions to E&V technology, much of which is embodied in documents attributed to the E&V task. These include documents such as the *Requirements for Evaluation and Validation of Ada Programming Support Environments* [51], [52], which provides direction for developing the required technology. From contracts sponsored to further the development of this technology, an initial version of an Ada Compiler Evaluation Capability (ACEC) has been released [44], and it may soon be used for "formal" DoD evaluations

[41]. First versions of the *E&V Reference Manual* [50] and *E & V Guidebook* [49] have also been released. The former provides a basis for an APSE evaluation scheme, and the latter provides checklists and references to E&V tools in current use.

The E&V team distinguishes between evaluation and validation. Evaluation is a measure of "goodness," which can be determined only within a specific context. Validation, on the other hand, is the determination of conformance to a specified standard [52]. This research was concerned only with software evaluation technology, determining how "good" a piece, or a collection, of software is for a particular purpose.

Although the E&V team has made a more concentrated effort than any other organization in advancing the state of APSE evaluation technology, others have also made significant contributions. Many of these have been reported in *Ada Letters* [2], [17], [75], [149], *Communications of the ACM* [25], [35], [66], the proceedings of Ada conferences [92], [129], and The Ada Companion Series of books published by the Cambridge University Press [104], [115], [123]. Others have been work sponsored by the DoD [33], [61], [161] and by the Ministry of Defence in the United Kingdom [124]. They have ranged from very specific areas, such as Ada runtime environments, to broad general coverage of Ada software evaluations. The general body of knowledge on software evaluation [59], [80], [97], [100], [121], [126], [171], and especially performance evaluation [60], [85], [132] is important in this research as well because much of it applies. Likewise, selection methods in general

[3], [43], and compiler evaluation in general [8], [79], [99] are also areas which contribute to the research. The performance evaluation of compilation systems is currently the area of most interest in APSE evaluations. However, although a good deal of work has been done to determine appropriate ways to summarize such information, no work has yet been done to integrate this type of evaluation information with other more subjective evaluations through a mechanism such as a decision support system.

The Decision Support System (DSS)

A decision support system is a software system which aids the user in making a decision of some type. The software itself does not make decisions for the user, but rather leads the user through the decision making process. It is an effective mechanism for maintaining consistency in the decision process [94]. In practice, a DSS is most often used on management problems, while expert systems are generally used to assist with decisions on technical problems. The proposed research will design a system which will assist a technical manager in making a decision, using evaluation information which is technical in nature. Hence, both managerial and technical points of view must be considered, and it is useful to compare the DSS and expert system approaches.

DSS and expert systems are both knowledge-based systems. In other words, rather than dealing with just a database containing perhaps unrelated data, they also deal with a database of "organized" knowledge referred to as a knowledge base. The main difference is

that expert systems are usually used to make decisions on well understood technical problems while DSS are used for more ill-structured management problems. Thus, an expert system typically comes up with one recommended solution to a problem, whereas a DSS may suggest multiple solutions or just provide useful information for the decision maker without making any recommendations [54], [81], [89], [105], [135], [144], [159], [173].

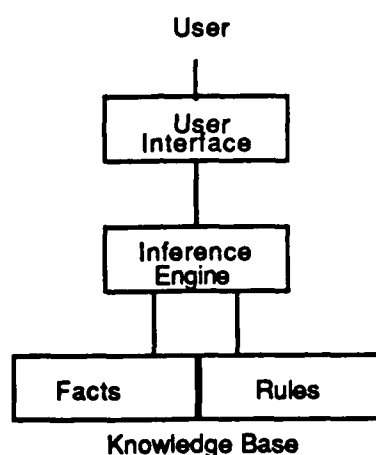


Figure 2.3 - The components of an expert system [105]

Either type of system is usually described in terms of independent, distinct components. Figure 2.3 illustrates the expert system components. They are generally considered to be the user interface, the knowledge base (containing both facts and specific rules for how the facts in this particular application domain may be manipulated), and the inference engine (containing general problem-solving procedures, usually containing heuristics). An explanation facility is usually also considered to be necessary, and an acquisition

facility is required for getting more knowledge into the knowledge base [105], [159].

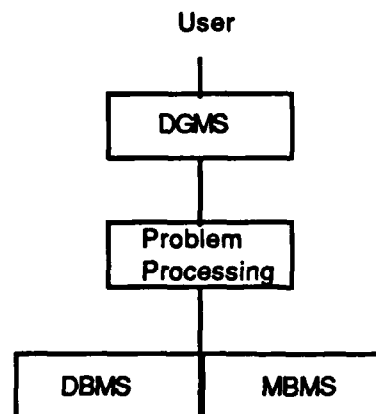


Figure 2.4 - The components of a DSS [155]

In a DSS (see Figure 2.4) the components are generally considered to be the dialog generation and management system (DGMS), the database management system (DBMS), and the model base management system (MBMS), with an implicit problem processing system controlling the system processing. The DGMS controls the dialog with the system user. The DBMS contains data, and the MBMS contains the analysis and interpretation facility for the system which can be based on statistics and algorithms or it can use data abstraction models [98], [112], [134], [135], [144], [155].

In comparing the DSS with the expert system, the components can be seen to relate directly with each other. It is not unusual for this similarity of form to be found in different disciplines (even in ones which seem totally unrelated) [77]. The user interface of the expert system is essentially the same as a DGMS of the DSS, the DSS

DBMS is much like the facts of the knowledge base of the expert system, and the MBMS of the DSS contains general problem-solving knowledge which may use heuristic techniques, much like the rules of the expert system knowledge base. The problem processing system of the DSS is more or less comparable to the inference engine of the expert system. While experts in either area would undoubtedly argue for shades of differences in the use and implementation of these components, the point here is simply that the structure of either could be used to describe the system which was designed in this research.

The ASSIST system will be described as a DSS because its purpose is to assist a decision maker and because it will be working with an area of knowledge which is not yet well understood (once evaluation technology matures, an expert system may become more appropriate). Hence, it will not necessarily be expected to provide just one recommended answer for the decision maker. However, principles of expert systems may also be applied. For example, DSS literature does not place particular emphasis on the system giving reasons for its recommendations, but expert systems are usually described as having this capability. Perhaps such information is just understood to be a part of the DSS because the ill-structured nature of the problem makes following the reasoning essential for the decision maker. Whatever the case, ASSIST makes the system reasoning clear.

Another area more specifically addressed by expert system theory than by DSS theory is the acquisition facility for getting

information into the system knowledge base. This facility is provided by a separate system component, and it is clearly needed in a DSS as well as in an expert system. ASSIST contains such a facility, and it both acquires data (facts) for the ASSIST database and knowledge (organized knowledge about how to deal with the data in the database) for what is referred to as the ASSIST knowledge base.

In designing a system, both DSS and expert system philosophies stress that the components must be as separate and independent as possible. They also claim that design and implementation cannot be separated [89], [105], [135], [144], [159]. The experts in these fields generally seem to think that these ideas are very different from the conventional wisdom of computer science and software engineering. However, although the artificial intelligence community was the first to discuss software development in these terms, current software engineering philosophies are not all that different.

System Design

The "conventional wisdom" of software development, rejected by DSS and expert systems experts, is based on the early waterfall model of software development (see Figure 2.5). It is, however, no longer the conventional wisdom for software development. The waterfall model serves as a good way to describe the various activities which comprise the development process, but techniques based on it have not proven to be particularly effective [47], [127].

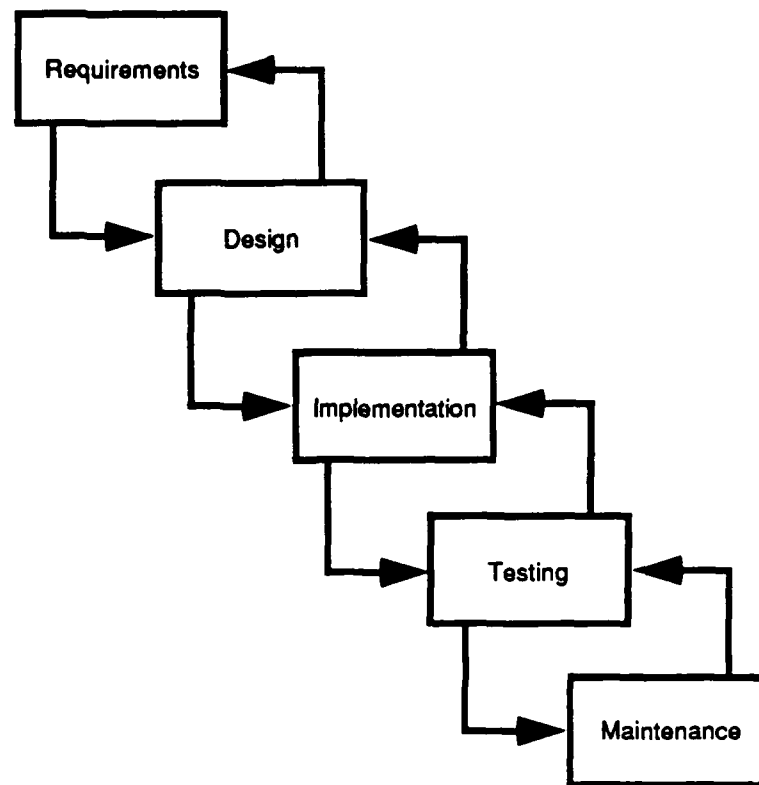


Figure 2.5 - Waterfall model of software development [6]

Two other related techniques, often referred to as evolutionary development and rapid prototyping, are currently recognized by many software engineers as more effective under most circumstances [5], [6], [7], [31], [103], [147]. These techniques use what Boehm calls a spiral model of development [19], where the design and implementation of a system are done incrementally. They are not mutually exclusive techniques. Although the spiral model does not really change the activities involved in software development, it does change the ordering and interrelationships among those activities.

Along with the maturation of philosophies on software engineering development techniques has come an increased

recognition of the importance of module or component independence. Current wisdom stresses the development of interfaces which permit outside components to know how to use an isolated component without knowing anything about its implementation [22], [30], [119]. This idea, combining the use of abstraction and information hiding, has been emphasized by newer languages (such as Ada) which provide built-in capabilities to separate component interfaces from implementations. Thus, current software engineering philosophy is in line with the one used by DSS and expert systems developers because both stress the importance of independent components developed using a process which mixes design and implementation in incremental steps.

In this research, prototyping and evolutionary development were used with the spiral development approach, and the main components (at the highest level of abstraction) were similar to those described for a DSS, with the addition of a knowledge acquisition component. However, the DGMS is called the user interface, the MBMS is called the knowledge base, and the problem processing system is called the decision logic to make the names more descriptive for a wider audience. The DBMS (made up of facts) and the knowledge base together are what an expert system collectively refers to as the knowledge base.

The object-oriented paradigm has become very popular among those who are trying to use the maturing ideas of spiral development and component independence. Techniques based on it are particularly well suited for developing independent components, and

they tend to structure a system in a way that makes incremental development relatively straightforward. This is probably why object-oriented techniques are most often used with Ada, despite the fact that Ada is not really an object-oriented language. Ada developers are trying to find the most powerful combination of techniques which will support the principles of software engineering [21], [30], [40], [42], [58], [71], [93], [127], [164], [166], [170].

Both the terms object-oriented programming and object-oriented design are commonly used to describe the process from design through implementation of a system using the object-oriented paradigm. For simplicity, we will refer to this process as a design technique. The strength of this technique is that it brings together the strengths of the function-oriented and data-oriented design techniques which have been most frequently used in the past [1], [18], [20], [26], [127], [166].

Not only was the object-oriented design technique appropriate for use in this research, but an information organizing technique called hypertext was as well. In a DSS, the organization of the knowledge base forms the basis for the decisions supported by the system. Hence, the better the technique used to organize this knowledge, the more effective the system can be. Hypertext is a technique which no longer requires databases to be structured in traditional, sometimes limiting, forms. The principle of the hypertext paradigm is that any information can be easily linked to any other associated information. No particular form of structuring (linear, hierarchical) is required. This makes hypertext a good vehicle for

implementing knowledge structures such as semantic nets [28], [32], [38], [39], [67], [73], [128], [140], [152], [169], [170].

This concept of hypertext is not entirely new, but it has not been possible for the concept to be put to practical use until it could be implemented in a computer system [28], [38], [39]. Computer processors are now fast enough that such implementations are possible, and a number of hypertext products are available for various personal computers. The hypertext technique can be used very easily in conjunction with the object-oriented design technique, providing a suitable vehicle for use in designing a DSS.

In this research effort, a hypertext implementation, HyperCard, was used for developing the system prototype. It has built-in facilities supporting each of the user interface, the knowledge base, the decision logic, and the acquisition facility subsystems of the DSS, called ASSIST. This permitted the research work to concentrate on the real issues. The built-in HyperCard user interface facilities made ASSIST user interfaces easy to develop. It also provided a capability for data and knowledge acquisition. This may not be a good facility for acquiring the huge amounts of data which would be required in a fully functional DSS, but it was adequate for the limited amount of data required for this research. Although the HyperCard database facilities may or may not be particularly efficient, efficiency is not required of a prototype, so this was not a concern of this research. HyperCard permitted a frame-based representation (sort of a semantic net with procedures attached) of the "rules" in the knowledge base. This was essentially accomplished via the object-

oriented nature of HyperCard, where the objects (cards, fields, buttons, etc.) formed the nodes of the net (the frames), and the objects' methods were the attached procedures. The design effort for the decision logic was also accomplished via the object-oriented nature of HyperCard's programming language, HyperTalk.

With this background established, it is time to turn our attention to the development of the object of this research, the DSS called ASSIST. The first major step in this process was defining ASSIST's requirements, a step which included the development of the first version of the ASSIST prototype. This requirements definition is detailed in the next chapter.

3. System Requirements

Although the design of ASSIST followed the prototyping model, the activities to be accomplished were still the same as those defined by the waterfall model. The difference is that when prototyping, one activity is not completed before the next activity begins. In fact, parallel work on multiple activities permits them to receive feedback from each other.

Even before work started on a prototype or a requirements document, input was solicited from the user community. Once this was obtained, activity turned to the development of system requirements. The prototype was started soon thereafter, and it played a large role in the continuing refinement of the requirements. This chapter details the process used in developing the system requirements, including the contributing prototype development. The requirements themselves are contained in a separate document. This requirements document, and the other documents prepared as a part of the process of defining the requirements, are contained in appendices.

Introduction

In developing the system requirements, it was important to consider that a software tool should be evaluated not only as a tool which provides a particular function, but also as a part of the environment in which it resides. Does the tool produce a "quality" product in a "reasonable" length of time using an "acceptable" amount of resources, and does that product work "well" with the

products produced by other tools in the environment? The question can be answered only after the attributes given in quotes in the previous sentence have been quantified, and this can be done only within a context which defines how the software will be used.

As an example of how difficult software assessment can be, consider the way compilers are often evaluated. A particular benchmark program is used. The program is compiled using Compiler A, and the execution time of the resulting object module is measured. This is then compared with the execution time of the same program after it has been compiled on Compiler B. Compiler A is considered better than Compiler B if the program executed faster using Compiler A.

However, the resulting assessment is faulty in numerous ways. In the first place, all benchmark programs will probably not give the same relative results using the same two compilers. Furthermore, the only attribute which has been measured is execution performance, and the results may or may not have been biased. Were both programs run on the same computer system under the same conditions? Does either compiler have optimization capabilities, and if so, what types and were they being used? Was it a multiprocessing system, meaning other programs could have been executing at the same time and competing for the same resources? Were the execution times measured as elapsed time (such as on a stop watch) or was a system function used to measure only the amount of time the program was actually executing (not counting other "administrative" functions which may have been going on as

well)? And if a system function was used, what are its performance characteristics? These are only some of the numerous questions which must be considered in order to arrive at an unbiased evaluation of just the execution performance for that particular benchmark program. And even if one were to have the results from a very comprehensive suite of compiler performance evaluation software (an elusive goal in itself), the only attribute of the software that would have been evaluated is still execution performance [33], [48], [50], [52].

In general, basing an evaluation on only one aspect of any piece of software is extremely unfair and misleading. Tool A could execute a little bit faster on a given benchmark than Tool B, which does the same job. But Tool A could also use ten times the amount of space as Tool B while it executes. So if company X were to select Tool A based on its execution speed (on the given benchmark) alone, it could be making a big mistake. In the first place, the execution speed that was measured may not be indicative of the relative speed for the application of interest. Furthermore, the space Tool A uses may not create a problem until after Company X has committed to a time schedule and started development of a new product. This could result in a real crisis when Company X realizes it cannot do the job with Tool A, and purchasing Tool B now not only causes a schedule delay but also a funding problem. Of course, this whole dilemma could have been prevented by getting a proper perspective on evaluation attributes of the software at the time Tool A and Tool B were originally compared.

Software has not been completely evaluated until all of its important characteristics have been identified, their relative importance has been determined, each characteristic has been assessed, and the results have been combined to form a comprehensive profile. However, such a profile cannot be compared with profiles from other software unless the profiling is consistent, always measuring the same things and reporting the results in the same form. Furthermore, even if a tool has been assessed as a quality tool in its own right, this does not necessarily mean it will be acceptable within a specific development environment and for a specific application. If it does not work well in concert with the other tools in the application environment, it may be awkward, annoying, and simply unacceptable in that particular context.

Achieving a comprehensive and consistent evaluation profile is a very difficult task. It goes far beyond the performance evaluation example given above for compilers. Yet most aspects other than performance evaluation of software are much more subjective, and hence much more difficult to quantify. This is why much of the current work on APSE evaluation has been in the area of performance evaluation of compilers. Although this is obviously a very important area in evaluating an APSE, it does not even address the complete evaluation of a compilation system, let alone provide any help in assessing other APSE components or the APSE as an entity itself.

Current software magazines attempt to make comprehensive evaluations for some types of software, but they typically use one

reviewer who only evaluates six or seven broad categories on a subjective rating scale [118], [122], [151]. This is probably better and more consistent than just relying on vendors' advertisements or word of mouth for assessments, but it certainly lacks comprehensiveness. Furthermore, it is only based on one reviewer's concept of how the software will be used, which may not match other uses well at all. Decisions made concerning the purchase of critical and expensive software must be based on something better.

ASSIST must deal with all of these difficulties in an automated manner. It is not its purpose to do actual software assessments, but rather to use assessment information provided from other sources. This information can be entered into the database, and then it is up to the program to determine how to deal with the information. ASSIST must be able to make comparisons of evaluations of multiple implementations of any particular type of software, and it must also be able to deal with multiple assessments, possibly conflicting, of the same software. This automated approach cannot assume that all of the information in its database is valid and unbiased, but it must use what it has available to give the best possible recommendations. It must work on the principle that the larger the number of assessments of a particular software implementation, especially if they come from different types of sources, the better the recommendations that can be made about that implementation. It is also important that any recommendation should include information concerning the number and sources of evaluations used so the

decision maker has the proper perspective from which to make a decision.

Questionnaire

To get input from the user community, a questionnaire was developed (see Appendix A). It contained questions aimed at determining how a decision maker should interact with a DSS such as ASSIST and what type of recommendations the system should provide to the decision maker. This questionnaire was distributed to members of the E&V Team, as well as to a number of other individuals both in government and in industry. The individuals were chosen in an attempt to get relatively quick responses from both technical and managerial viewpoints.

Seven questionnaires were returned out of the 30 that were distributed. Although this was not a large number, the responses indicated that the questionnaire served its purpose. The varied backgrounds of the respondents resulted in two distinct perspectives on the issues involved. The technical perspective showed a concern for technical correctness and completeness, while the managerial perspective showed an interest in ease of use and simplicity of results. This provided insight into the issues which had to be balanced in the development of ASSIST, and it was a valuable beginning. Many of the specific suggestions on the questionnaires were also useful in later stages of the ASSIST development.

The Requirements Document

The first goal in the development of any system which will be more than just a "toy" is to develop a set of system requirements. A very good guide in developing requirements is ANSI/IEEE Std 830-1984, *IEEE Guide to Software Requirements Specifications* [12], and this standard was used as the model for a requirements document for ASSIST.

It was not expected that the document could be written without doing some work on a prototype, but it was possible to rough out the form of the document first. Then some preliminary and rather general ideas were written out as the first form of the system requirements. It had already been decided that ASSIST would have the general form of a DSS, and four subsystems were identified, along with their general functions.

The Subsystems

Figure 3.1 illustrates the structure of the ASSIST subsystems. One is the User Interface Subsystem. The function of the user interface of any system is rather well-understood, although not necessarily easy to develop. The user interface is important to ASSIST because it may well determine whether the decision maker will ever use ASSIST. However, its development is not an important part of this research. The research only defines its requirements.

Another subsystem is the Knowledge Acquisition Subsystem. This is the subsystem to handle the conversion of all input into the proper form for both the database and the knowledge base of the

system. The database contains all the evaluation information collected from various sources. This data could be entered in many forms, and it is up to the Knowledge Acquisition Subsystem to convert the data to the form to be used in the system database. For large amounts of data it makes sense to input it from a file, but an interactive capability is desirable also.

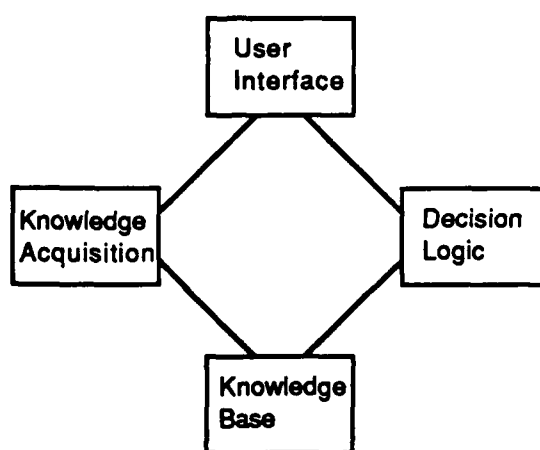


Figure 3.1 - The structure of the ASSIST subsystems

The knowledge base contains the more "organized" information which determines how the system works with the data in the database. This knowledge was one of the key parts of the research, but the important part was the form of the knowledge, not how it gets converted into that form from the user input. Hence, the design of this subsystem was not of great concern, and only its requirements were defined.

The next subsystem to consider is the Knowledge Base Subsystem. This subsystem performs all of the actual storing and retrieving of information to and from both the database and the

knowledge base. Accessing the database is another straightforward function, typical of any commercial DBMS. On the other hand, accessing and using the knowledge in the knowledge base is a key issue in this research. Hence, only the knowledge base requirements of this subsystem were defined.

The final subsystem, the Decision Logic Subsystem, is the one of most interest. This is the subsystem which determines what action ASSIST will take at any particular time. It leads the decision maker through the process of specifying the type of software to be selected and the criteria to be used in the selection process. The design of this subsystem and how it interacts with the knowledge base was the main thrust of this research.

A consideration of importance was that the subsystems should be independent of one another, and the only communication among them should be via well-defined interfaces. In particular, the Decision Logic Subsystem was not to be designed with evaluation knowledge "hard-coded" in it. It should be designed to take the appropriate action at any particular time, but it should rely on the knowledge in the knowledge base to determine how to deal with the evaluation information.

Once the subsystems were specified at this high level, more specifics were needed about how the system should work before more detailed requirements could be written about it. This is what a prototype does best, so it was time to turn away from writing a document and start refining ideas via a prototype.

The Prototype

The next job was to determine how much the prototype should do initially. Developing a prototype entails doing a little bit of each of requirements, design, implementation, and testing, on a small scale. The trick is to keep the scope appropriately small so important decisions about the system are not made too early, but at the same time to make the prototype capable enough to be able to firm up ideas on how the system should work.

For ASSIST, the important thing to be determined by the prototype process was how the user would specify both the type of software to be selected and the criteria to be used in the selection process. However, the prototype software had to accomplish more than this. It also had to provide a system with a high degree of usability because feedback from prospective users would be a valuable way to determine the viability of the ideas incorporated into the prototype.

Context Specification

At the beginning of the prototype development, the important user specification process had to be defined, starting with an organization for the APSE evaluations. The DoD's E&V Team has already put considerable effort into this area, and the result was not just a simple taxonomy. Rather, several views were considered. The E&V reference system contains views of software evaluation by life cycle activities, tool categories (such as project management system, compilation system, etc.), attributes (such as efficiency, reliability,

maintainability, flexibility, etc.), and functions (categorized under transformation, management, or analysis). In addition, it considers issues of importance when assessing an APSE as a whole. These taxonomies provide several dimensions to the evaluation process [48], [50].

Based on this work, multiple dimensions were incorporated into the prototype design. The user sets the context by choosing the scope of the software to be selected (an individual tool, a tool set, support for a particular life cycle activity, or a whole APSE), and then choosing the specific type of software within that scope (a debugger, a compilation system, support for requirements definition) if it is a confined scope as opposed to the whole APSE. By having the user select each of these elements from a number of choices, rather than typing them in, the system can be sure the type of software is specified using the system's terminology. The user does not have the burden of trying to type terms which must be both spelled correctly and understood by the software.

The Importance of the Application Area

Next, the user chooses the application area in which the software will be used. This is not an absolutely necessary piece of information for ASSIST to have, but its specification serves two purposes. First, the user is reminded that the application area may influence the things which are most important in selecting the software. Second, the system can use the application area information to provide appropriate suggestions to the user. This not

only makes ASSIST easier to use, but it also gives the user more confidence that important criteria are not being overlooked in the selection process.

Choosing Features and Criteria

The next step was a focal point of the research and also the most difficult step to define. Not only is it important for the user to specify characteristics to be evaluated in the selection process, but it is also necessary to know the relative importance of each. Furthermore, sometimes it is important to be able to require certain absolute characteristics of the software. For example, it may be critical to the success of a hard real time system that the Ada compiler permit the specification of machine representations. In this case, the user may want to be able to indicate that only compilers which have this ability should even be considered as the software to be selected. A separate issue is how well the compilers use this characteristic when they have it [59], [61], [80], [97], [104], [106], [161], [171].

Thus, it was decided to specify the importance of an absolute characteristic (referred to as a feature) and the importance of a relative characteristic (called a criterion) separately. Keeping the two concepts separate helps the user to define the selection process more precisely and with less confusion.

The actual process of choosing both features and criteria are handled in much the same manner by ASSIST. The main difference between the two is that the knowledge base provides different ways

for the system to deal with them. In addition, the user is not required to specify any particular feature as important, but at least one important criterion must be specified before comparisons and recommendations can be made.

It is not desirable to limit the user to a maximum number of features or criteria which may be specified. Although any DSS restricts a user to some extent, it is best if the restrictions are not perceived by the user [139]. On the other hand, the specification process is probably most meaningful to the user when the number of characteristics to consider at any one time is limited. It was decided that ASSIST would address this issue by providing the user with a limited number of choices at any particular level of specification for any particular characteristic. However, the number of levels used would not be restricted by anything except the amount of knowledge in the system knowledge base.

The prototype is set up to provide lists of suggested features and criteria which are usually important for the particular application area. The user could choose as many as desired from these given lists, and in the later version it would also be possible to add others. The concept of different levels of specification was also suggested in the first prototype, but it was not implemented until the later version.

Weighting the Features and Criteria

After the most important features and criteria have been chosen, the user must also have the ability to indicate their relative

importance. The ASSIST prototype provides suggested weights for each characteristic selected. These range from 1 to 10, where 10 indicates most critical importance. However, the user must also be provided with a mechanism for being able to change any of the weights, if desired. The prototype provides this capability.

The Calculations

Once everything has been specified, ASSIST must use this information, together with the evaluation data in its database, and massage it in accordance with the knowledge in its knowledge base, to provide recommendations to the user. This process involves using the system's knowledge of how to assess the characteristic to determine a numerical rating for it.

The first version of the prototype used a simple rating scheme because the knowledge base was not developed to any extent. Very few features or criteria were supported, and the data in the database had to indicate that the software with respect to a criterion was rated poor, satisfactory, or good. No other information was allowed.

The prototype awards a rating of 0 if the software is poor with respect to a criterion, or if the criterion has not been evaluated for that implementation. It awards a 1 if the software is rated satisfactory, and a 2 if it is rated better than satisfactory with respect to the criterion. For features, a rating is 0 if not present or not reported as present, and a 2 if it is present.

Determining how to combine individual ratings to arrive at an overall rating is discussed at length in the literature on decision

theory. One area of concern is whether the individual characteristics are independent of one another. Decision theory has some straightforward ways to deal with independent characteristics. Depending on how deeply one gets into this theory, it can also become much more complicated. However, no particular complicated form seems to be any more universally accepted, nor does any produce consistently better results, than the simplest forms which assume that one can use straightforward linear additive functions for independent characteristics [64], [74], [136], [143], [165], [172].

ASSIST uses this method of weighted averaging with linear additive functions for features and criteria. The first version of the prototype assumes that all features and criteria are independent. The second version looks more closely at the dependency relationships among the characteristics.

To calculate a weighted average, the numerical rating (indicating poor, satisfactory, or good) for a characteristic is multiplied by the characteristic's weighting factor (specified by the user). This results in a weighted rating. All the weighted ratings for all features are then averaged to arrive at an overall feature rating. The same process is used for an overall criteria rating.

These overall ratings for the features and criteria determine the final rating for a particular software implementation in the database. The prototype treats the overall feature and criteria ratings with equal importance in the decision process, but this is not necessary. It was decided that future versions should weight the overall feature and criteria ratings to establish their relative

importance. Users will be able to change the default parameters used to weight these ratings as desired.

The Results

The final results of the calculations must be reported to the user in some fashion. As with the other information presented to the user, meaningful chunks of information are to be provided at a time [15], [32]. These are to be presented to the user at different levels of detail, starting with the highest level containing little detail. The user can then request more and more details, looking only at as much information as desired.

The first prototype implements only the highest level of detail, which is the presentation of recommendations to the decision maker. After some consideration, it was decided to present up to three lists of recommendations to the decision maker, as applicable. The first list contains acceptable software implementations meeting the user-specified requirements, sorted in order according to rating. Implicit in deciding whether an implementation gets on this list is a determination of what constitutes an "acceptable" rating. Since 1 is a satisfactory rating in the prototype, it was decided that any final rating above 0.9 would be considered acceptable. However, in the later version, this would also be available to the user as a parameter which may be changed.

The other two lists are intended as information which can give the decision maker a perspective of the recommendations, indicating the breadth of information in the system database. The first list

presents all implementations which were considered but did not meet the acceptable rating, while the second list contains all implementations which were not considered because they did not contain at least one feature which was specified by the user to be absolutely essential.

When fully implemented, ASSIST will provide more detailed levels of information concerning the recommendations given and the means of arriving at these recommendations. The calculations, as well as the parameters used, will be fully explained. The user will also have the chance to change the calculation parameters at this time. At the very lowest level of detail, chunks of the actual data in the database will be available to the decision maker for review.

The User Interface

More than just accepting user input, the prototype also had to establish a reasonably good user interface for use by the reviewing decision maker. The goal was to establish an interface which would not change significantly as ASSIST changed from a prototype to a production system. This meant that the full spectrum of system requirements had to be considered in developing the user interface.

The prototyping process shows its spiral nature at this point. First, a general set of requirements was generated, followed by some work on the prototype requirements. This fed the prototype design and implementation. However, before getting very far on this design and implementation, the system requirements had to be reexamined and developed a little further to establish the scope of the entire

system. Then the prototype user interface design could be continued.

This user interface design had to account for a number of peripheral activities which are important to the usability of ASSIST. The system must have:

- 1) the ability to backtrack to the previous window,
- 2) a graphical browser permitting the user to "see" how the current activity fits into the "big picture" of the program,
- 3) an ability to review the reasoning process at any particular time during program execution,
- 4) an ability to print results,
- 5) an ability to save and retrieve the specifications which have been input by the user, and
- 6) an on-line help system.

It was decided that each of these activities should be available to the user at any time during program execution. With some systems, this must be accomplished by requiring the user to remember obscure key combinations. However, with HyperCard this was accomplished rather easily by defining icons (small pictures) to represent each of these activities. The icons could then be on the screen, available for selection at any time, but they would use very little of the space in the current window.

The first version of the prototype did not fully develop all of these activities. However, backtracking was easy to implement by popping an internal stack. The on-line help system was a natural outgrowth of the information which had to be composed to explain

the user steps necessary to complete the specifications for the software to be selected. So on-line help was implemented. A simple graphical browser was also implemented as a method for the user to move around in the specification process, allowing specifications to be changed at any time.

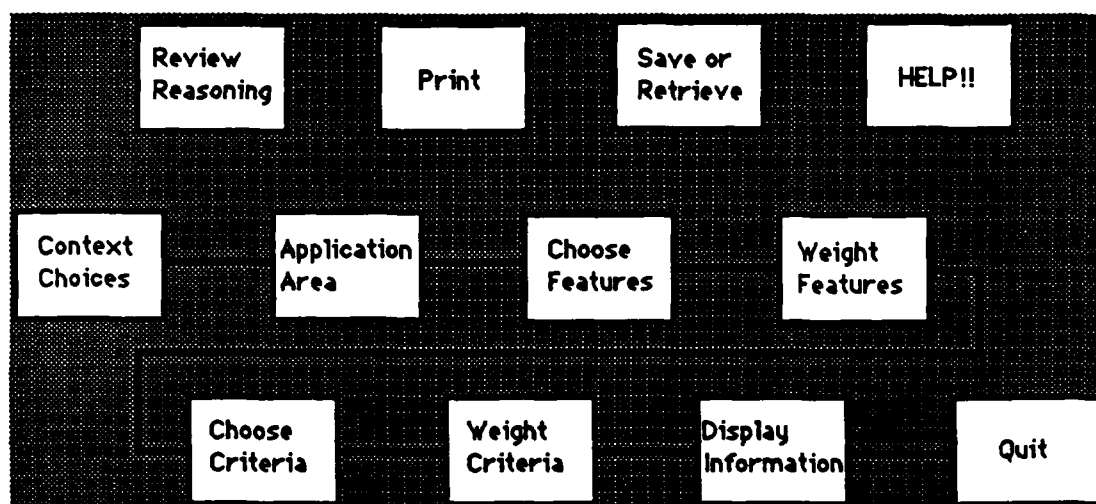


Figure 3.2 - Relationships among ASSIST activities

Once the user interface was completed, the user specification process, described above, was integrated with it. Figure 3.2 is a graphic of the relationships among the various activities of ASSIST. No attempt was made to do anything beyond providing the user with the simplest abilities to make specifications for the software to be selected. A simple database and knowledge base were defined, but only as much as necessary to permit a user to execute the program. However, the complete process of specifying the software to be selected was represented, and the prototype provided the user with recommendations based on its simple database. The most important

aspect of the prototype was accomplished because the basic system execution process was defined.

Reviewing the Prototype

Once this simple first version of the prototype was completely developed and tested, a number of individuals were invited to review it, including all who had responded to the initial questionnaire. As with the questionnaire respondents, the backgrounds of the reviewers ranged from a high level of technical involvement in software evaluations to a non-technical management orientation. A comment sheet was provided to each of the 11 reviewers (see Appendix B), inviting suggestions about each of the prototype activities.

The feedback from the reviews was very positive overall. Most of the concerns and suggestions were in areas in which the prototype was incomplete, and they pointed to the lack of sophistication of the knowledge base. The most negative response was from one technically-oriented skeptic who doubted that any computer system would ever be able to provide adequate technical information for a decision maker. This, of course, challenged the thesis of this research and made it clear that much work still remained. Overall, the comments provided confirmation that the development was progressing in the right direction.

Back to the Requirements Document

The development of the first ASSIST prototype accomplished its purpose. The whole idea was to develop just enough to make it possible to go back and refine the system requirements in much more detail. This was possible after the first prototype development because the entire process of the program execution was much more clear. The peripheral functions of importance had also been determined, and detailed requirements for them could be put into writing as well.

One of the objectives in developing detailed requirements is to be sure they can be tested once the system is completely developed. A system test plan should be developed at some point, reflecting test objectives which will test the accomplishment of each of the requirements. Test descriptions must then specify exactly how each of these objectives will be tested. It was decided to develop the test plan and accompanying test objectives and test descriptions in conjunction with the requirements document. This would help to refine the requirements and to ensure that they are testable.

The Test Plan

The test plan was developed using the standard, ANSI/IEEE Std 829-1983, *IEEE Standard for Software Test Documentation* [11], as a model. The requirements to be exercised by each test objective were identified and cross-referenced to the requirements document. This made it possible to be sure of complete test coverage. Objectives for test scenarios were also identified, where test scenarios would be the

actual vehicle for exercising multiple tests at one time. Finally, test descriptions were developed for each test objective, further defining how the requirements could be tested and also providing feedback for refinements to the requirements. The descriptions of the test scenarios were also provided as a part of the test descriptions. This resulted in the documentation of everything testers would need to know to test the system except for specific inputs and specific expected test results. These specifics can only be determined after the system database and knowledge base are better developed, so they were left for future work.

All of the documents developed in this process of defining requirements are shown in appendices to this document. The Requirements for ASSIST is Appendix C, and Appendix D is the Test Plan for ASSIST. Appendix E details the test descriptions for each of the test objectives in the test plan, and it also provides cross-references to the requirements covered by each test. Appendix F provides cross-references from the requirements to the tests and also to the system design, which is described in the next chapter.

Once the requirements were developed to this point, it was time to shift attention once again and get into the meat of the system design. The full process of selecting evaluation features and criteria now had to be developed, and this was closely tied to the knowledge base. It was time for developing this knowledge which would be used in dealing with the various types of evaluation information. The next chapter looks in depth at the full extent of the ASSIST

design developed in this research. Once again, the prototype proved to be a valuable aid in the design process.

4. System Design

It is the system design which defines the actual structure of ASSIST. Although the requirements specified the structure of the subsystems, they went no further than that. They did not assume any particular structure to any of the subsystems. In this chapter, the structure of the Decision Logic Subsystem will be completely developed. The aspects of the other subsystems which are pertinent to this research will also be discussed, but, of these, only the structure of the knowledge base will be considered in any great detail.

First, the object-oriented design method used for the design will be discussed, along with the graphical symbols associated with it. Next will be a discussion of how HyperCard supported the object-oriented design techniques which were used. With this background, we can then proceed with a detailed discussion of each of the significant issues involved in developing ASSIST, as well as how each is addressed in the design.

As a part of the design process, the ASSIST prototype was enhanced. This provided additional insight into some of the more difficult areas, such as the design of the knowledge base and how the Decision Logic Subsystem would interact with it.

The two main areas which had to be addressed by the ASSIST design were:

- 1) how the software to be selected will be specified by the decision maker, and

- 2) how the recommendations will be made and presented to the decision maker.

Numerous issues had to be addressed within each of these areas. Each of these issues will be identified and discussed in detail.

Object-Oriented Design

Design techniques are a very hot topic in current software engineering research, and no particular technique has been universally adopted. There is not even an agreement on which design approach is the most effective. However, much recent work has pointed to the object-oriented approach as the most promising [23], [26], [40], [71], [127], [166]. Other work has shown a strong relationship among the object-oriented concepts, the knowledge representation concepts of frames and semantic nets, and the concepts of hypertext (or, more generally, hypermedia) [13], [38], [67], [73], [87], [169], [170]. Considering that the research described here is concerned with developing a DSS (which uses knowledge representation concepts) using HyperCard (an implementation of hypermedia concepts), the use of object-oriented design techniques was not only a reasonable approach, but a rather compelling one.

The design techniques used in this research do not strictly follow any one particular object-oriented method. They are most heavily influenced by Booch and his object-oriented method for designing Ada systems. However, they also show the flavor of HyperCard. This is not surprising since HyperCard was not only used for developing the ASSIST prototype, but it was also used for

producing all design documentation. The ASSIST design developed for this research may be found in Appendix G.

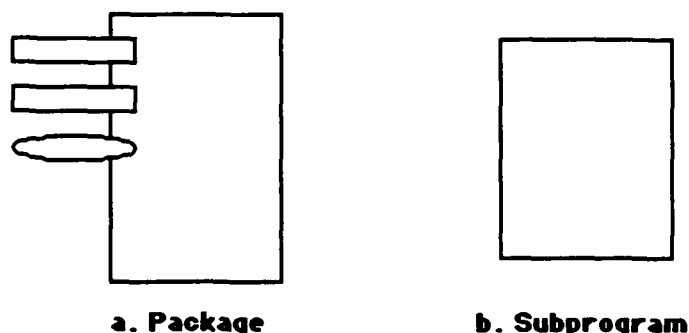


Figure 4.1 - Object-oriented design components [22]

The graphical symbols used are two of the four types of modules defined by Booch [22], illustrated in figure 4.1. The package symbol represents an Ada package construct. Of course, ASSIST could be implemented in a language other than Ada, such as in the prototype. However, the package construct still conveys the intended meaning, regardless of the language of implementation. The package represents a system object and its interfaces (indicated by the "windows" protruding from the side of the figure), which are the object's operations (or methods) and internal data objects. These interfaces indicate the "packaging" of a combination of elements which are resources available for use by other modules.

The subprogram symbol may represent an object which is either a procedure or a function subprogram. In either case, it is different from a package in that it only has one interface and it represents a single process rather than related elements packaged

together. However, either the package or the subprogram may define other packages or subprograms which are not visible in their interfaces. These would be their internal objects which are not available to other system modules for use.

Design Documentation ASSIST Diagram ↶	Description ASSIST will use a collection of evaluation data and user-given parameters in preparing and presenting advice to the user concerning APSE and other Ada software selection decisions. 🔍	
	Design Decisions The ASSIST subsystems will follow the general form of a DSS. The subsystems will be: User Interface, Decision Logic, Knowledge Acquisition, and Knowledge Base.	
	Design Level 0	Requirements 3.3.3

Figure 4.2 - ASSIST Design Documentation card

In the ASSIST design, each defined module has a Design Documentation card (see Figure 4.2) containing its number and description, as well as cross-references to both the system requirements and the system tests which apply to that module. The number indicates how the module fits into the breakdown of system

objects, with 0 representing the highest level object, ASSIST itself. The subsystems are then numbered (arbitrarily) 1 through 4 in the following manner:

- 1) Decision Logic Subsystem
- 2) Knowledge Base Subsystem
- 3) Knowledge Acquisition Subsystem
- 4) User Interface Subsystem

At the next lower level of abstraction are the objects which make up each subsystem. For example, the objects 1.1, 1.2, etc. describe subsystem 1 at a lower level of abstraction. In each case, a lower level is a more detailed definition of the higher level object.

If the module description requires more space than is available on the Design Documentation card, a button which looks like a magnifying glass is placed on the card. This links the card to an additional Description card. Additionally, any design decisions which were important to the definition of the module are also indicated on the Design Documentation card.

If the module is a subprogram and the algorithm it executes is nontrivial, a "PDL" button is present which represents a link to a Program Design Language (PDL) card. The PDL card details the algorithm used by the module in an Ada-like PDL [83] (see Figure 4.3). If the module has a lower level definition, a "Diagram" button is present, representing a link to a Design Diagram card.

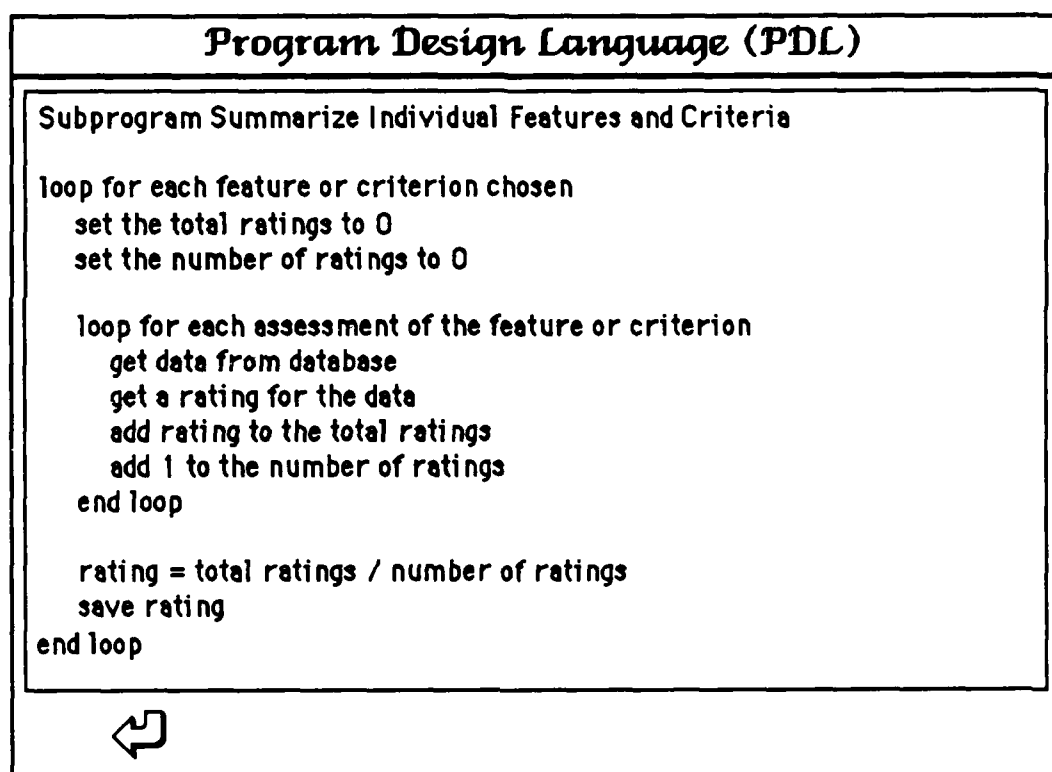


Figure 4.3 - ASSIST Program Design Language (PDL) card

The Design Diagram card (see Figure 4.4) contains a graphic of the objects which make up the lower level definition of the module, along with arrows which indicate dependencies among these lower level objects, if any (often referred to as a Booch "uses" hierarchy). An arrow originates in the body (inside) of an object and points to the top of the object it depends on (representing the originating object's use of resources via the interface of the object it depends on). The Design Diagram card always has a "Documentation" button linking it back to the corresponding Design Documentation card. If the diagram contains at least one package, the card will also have an "Interfaces" button linking to an Interfaces card which lists the visible interfaces to each package (see Figure 4.5). In addition, each

object in the diagram is itself a button which links to the Design Documentation card for that object. If appropriate, there is also a "Higher" button in the upper left corner which is linked to the Design Diagram card of the object at the next higher level of abstraction, and a "Top" button in the upper right corner linked to the Design Diagram card for ASSIST.

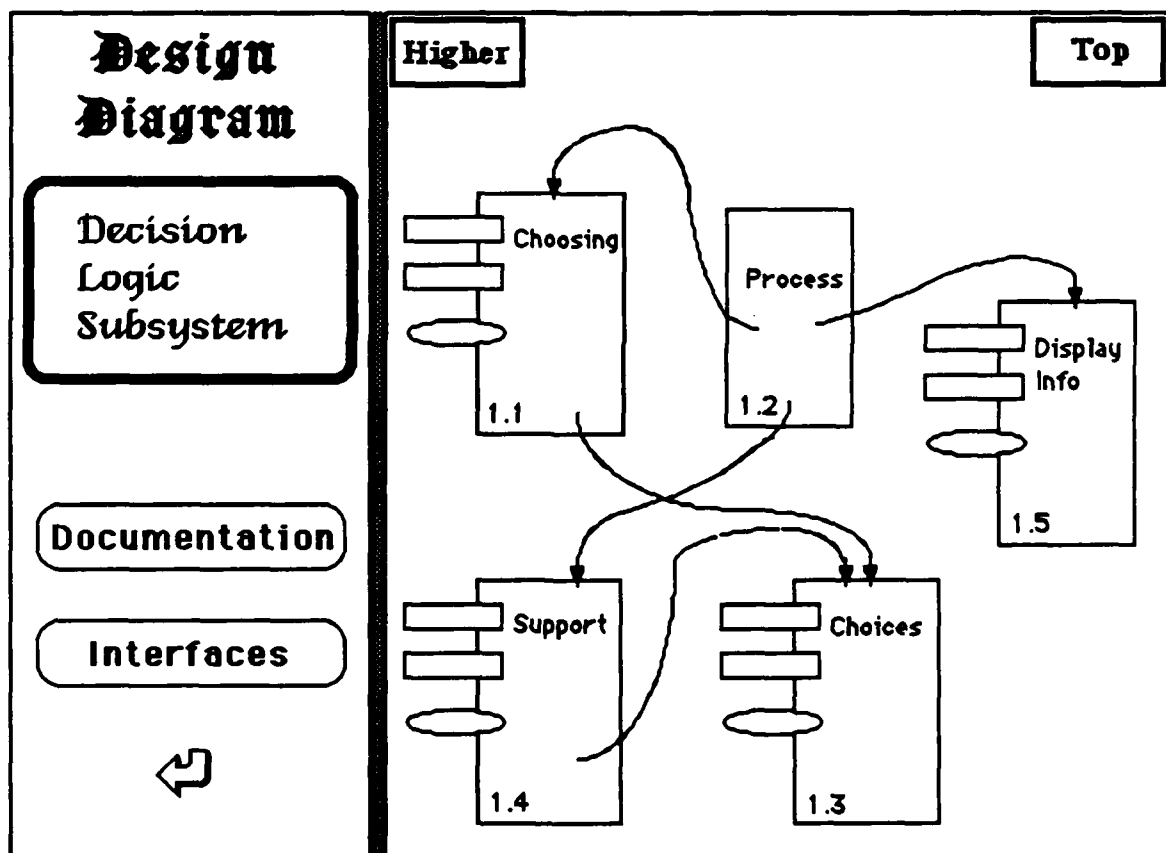


Figure 4.4 - ASSIST Design Diagram card

One button which is present on all the design cards is a return arrow. This button always links a card back to the previous card (via an internal stack mechanism). By using the buttons provided on

these design cards, it is possible to "navigate" throughout the entire system design when working with it in HyperCard.

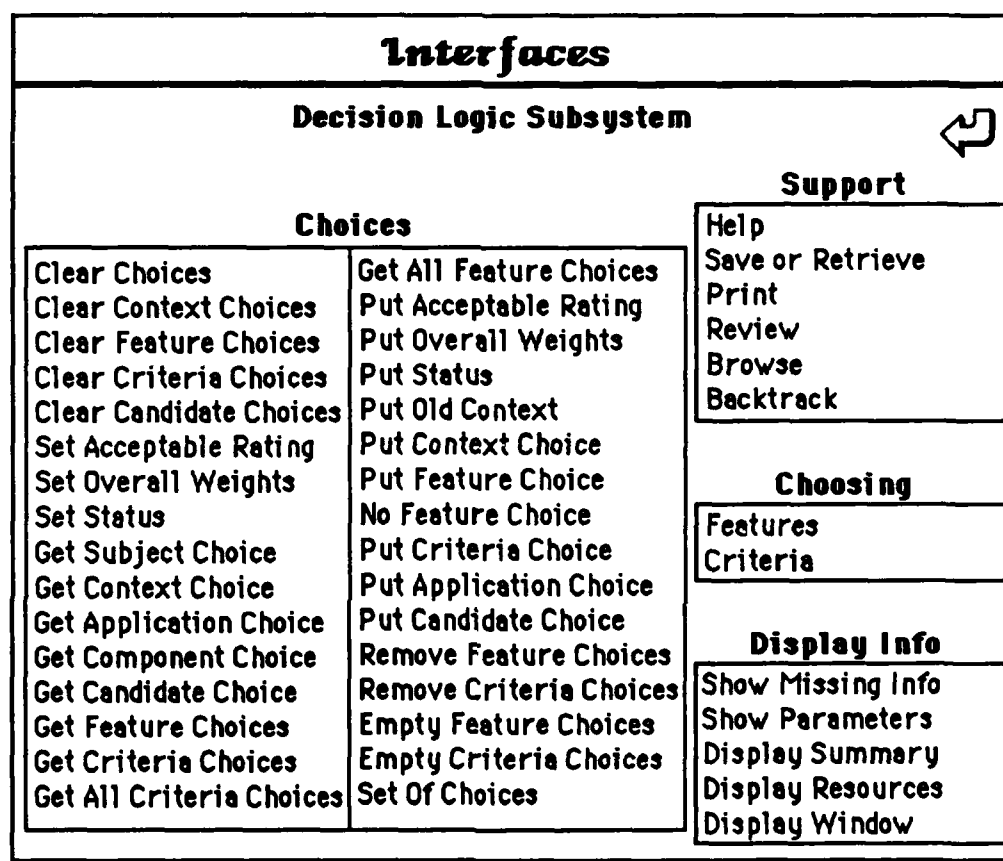


Figure 4.5 - ASSIST Interfaces card

The design documentation for ASSIST contains design information as described in the *IEEE Recommended Practice for Software Design Descriptions* [84]. However, the organization is not the same. Rather than being restricted to a linear progression through the information, the hypermedia ability of HyperCard has been used to permit easy access (via one mouse click on a button) to any card which is associated with the one currently being viewed.

Unfortunately, in Appendix G, we are restricted to the linear presentation of the design.

HyperCard Support and Drawbacks

As would be expected, the design and on-going development of the ASSIST prototype had a profound influence on the formal system design. The prototype design was, in fact, the basis for the system design. In developing the prototype, HyperCard provided valuable support in many ways, particularly in the area of graphical tools. However, HyperCard also presented some drawbacks which had to be dealt with. This section explains the basics of how HyperCard works and details those aspects of HyperCard, both helpful and not, which have undoubtedly influenced the ASSIST design. Care has been taken in order that any identified HyperCard shortcomings were not reflected in the formal system design specification.

How HyperCard and HyperTalk Work

The HyperTalk language, which is built in to HyperCard, is object-oriented and also very graphical in nature. The object support in HyperTalk is based on built-in hierarchical objects. These are, from highest to lowest, stacks, backgrounds, cards, fields, and buttons. A stack may contain any number of backgrounds, and any number of cards may be associated with each background, but each card has only one background. Fields and buttons are mutually exclusive and both are at the lowest level of the hierarchy. However,

a field or a button may be associated with either a background or a card [13], [72], [86].

It is important to realize that all of the objects, except the stack (which is the actual program), are associated with the built-in graphics of HyperCard. Hence, they are closely related to what the user sees on the screen. When working within the HyperCard application, the context is always that of a current card, which is usually what is visible on the screen (the only exception to this is when the screen image is "locked" via a HyperTalk command). A HyperTalk script (set of commands) is associated with each object, and the script is activated whenever a message is sent to the object. Sending a message to an object is much like a call to a subprogram in an Ada package. The name associated with the message determines the method (Ada calls it a procedure or a function and HyperTalk calls it a handler or a function) which is activated. HyperCard has many built-in messages, but HyperTalk also permits the definition of new handlers and functions, and these are automatically associated with new messages of the same name. Any handler or function may send a message to (call) any other handler or function within a stack [13], [72], [86].

The HyperCard objects form a powerful set of building blocks for an application, but the flexibility of HyperTalk is also somewhat limited. For example, although the objects may be composed in many ways, only the built-in objects may have scripts associated with them. This is important because it is the object and its associated script which bears the closest resemblance to an Ada

package. However, HyperTalk has no provision for separation of interface and implementation in the definition of an object. All of an object's script (all handlers and functions) is accessible to any script of any object. This puts the burden on the programmer to keep control of visibility, and this task becomes all the more difficult as the program gets larger.

In the ASSIST prototype, the process of having the user specify the important characteristics of the software to be selected was broken into a series of steps, and each step was associated with a card in HyperCard. Each of these steps was considered an object, and the methods for each object were put into the script of the object card. Buttons and fields were also associated with each card (they are visually on the card).

The various parts of the database and knowledge base were also structured on cards (ones which would not become visible to the ASSIST user), and the necessary methods for storing and retrieving information on each card were put in the respective cards' scripts. Messages are sent directly from one card to another to request information, and a handler (a procedure rather than a function) associated with that card returns information to the message originator. HyperTalk has the equivalent of Ada "in" arguments which can be sent to a handler, but it has no counterpart for Ada "out" arguments. However, using a mechanism called "the result", the object specified in a return statement (used in the same manner as the return in a function) is made available to the message originator [13].

The Subsystem Structure and HyperCard

Some problems remain in representing the structure of the subsystems. There seems to be no solution to them, at least not in the current version of HyperCard.

In the first place, it is impossible to separate the user interface from the other subsystems in HyperCard because the resources for the user interface are an integral part of HyperCard. This is not really a bad situation. If anything, perhaps it points to a part of building computer applications which will become much easier and more expected in the near future. If we start to see new software coming complete with its own powerful built-in resources for creating user interfaces, then we will be well on the road to accomplishing better productivity in software development through the use of more powerful tools. As to the design of ASSIST, since we are targeting the design for implementation in Ada, it cannot be assumed that such user interface resources will be available for ASSIST development. However, if they are (and this is quite possible since more powerful Ada development tools are being developed all the time), this can easily be accommodated by the given design. The User Interface Subsystem is conceptually nothing more than a package of resources anyway. It just needs to be available to both the Decision Logic Subsystem and the Knowledge Acquisition Subsystem.

Another structuring difficulty which is more bothersome is that the control logic of the Decision Logic Subsystem cannot be totally

separated from the subsystem objects. Ideally, the objects would be represented by cards, fields, or buttons, and the control logic would go in the script of the stack. However, much of the control logic is dependent upon the user choosing a button by clicking on it. It is usually most straightforward for the logic associated with the button click to go into the script of either the button or the card it is associated with. It is possible, though awkward, to put this logic into the stack script instead, but this creates another problem with no acceptable solution. HyperCard has a limit to the size of the script for any object, including the stack. The only way to get around this would be to create objects for the sole purpose of extending the space for stack logic, and this would defeat the purpose of separating the control logic in the first place. The purpose is to make the design structure clean and easy to understand, and this purpose would not be served by creating workarounds to an already awkward structure. Hence, the control logic is spread throughout the scripts of numerous objects in the stack.

Regardless of these difficulties encountered with the development of the prototype in HyperCard, the prototyping process served its purpose well. The system structure was better understood after implementing the various tasks in HyperCard, so the formal design now sports a better structure because of the prototype work. Much of the design was also easy to formalize because it followed the prototype design very closely.

Developing a Common Terminology

The first area which had to be addressed in the design process was the organization of the data and knowledge in the system knowledge base. This required an analysis of the form in which the data should be entered into the database as well as how the database would be searched. At least one transformation of the data would be required in getting it from the form in which it is collected to the form in which it would be used by ASSIST. The connection between the organization of the data in the database and the knowledge in the knowledge base also became a key consideration.

Data Transformation Issues

The data transformation could be done in two basic ways. The data could be entered into the database without transforming it from its many original forms. However, this would require an inordinate amount of transformation activity at a time when ASSIST must be doing already time-consuming database searches, as well as data manipulations and calculations. Hence, this was considered to be unacceptable.

The alternative is to transform the data into a more directly usable form at the time it is entered into the database. This requires the Data Acquisition Subsystem to be more sophisticated and take more time executing than with the first alternative, but it is much better for the system to take the time during a system management function than when the decision maker is trying to use the system. Using this alternative, it is also possible to take advantage of

interaction with the person entering the data to clarify items which are ambiguous, incomprehensible, or incomplete. Hence, this alternative was adopted for the design.

To determine the form in which the data should be stored in the database, the form in which the system would use the data had to be considered. The most straightforward way to access data is through the use of keywords, and this type of data search is supported by any DBMS. To use this most effectively, it was decided that the keywords should be the features and criteria which would be chosen by the decision maker as the software characteristics to evaluate. This would avoid any transformation between the user's choices and the topics to be searched in the database.

The type of data which would be stored along with each keyword also had to be considered. This would be the actual evaluation data which would determine the rating for the software with respect to a keyword (feature or criterion). The only practical way to make objective comparisons using both textual and numerical data is to transform all comparative data into a numerical form. ASSIST determines ratings for each software product by performing numerical computations (this process is described in detail later in this chapter). However, it is not necessarily desirable to transform all data into a numerical form to store in the database.

The decision maker will be entering preference data into the system to determine how to deal with the chosen keywords, and this data must be matched with data in the database. Some of this data will be easier for the user to work with in textual form (for example,

choices for the desired skill level of the user make more sense as *novice*, *intermediate*, or *expert* than as 1, 2, or 3). The numerical values to be used in the calculations cannot be determined in advance anyway because the user's choices usually determine the numerical values to be used (in the previous example, the skill level rating for software evaluated as requiring intermediate skills would be different depending on what level the user had chosen as desirable). Furthermore, when ASSIST is presenting recommendations, the decision maker may wish to see some of the details of the data used to arrive at the recommendations, and many of these details would make no sense in a numerical form. For all of these reasons, the data which accompanies each keyword should be in a form in which it is readily understood. From this form, the translation into the appropriate numerical form for the purposes of performing calculations will be simple and straightforward.

With these data transformation issues resolved, the details of how to perform some of these transformations can be completed elsewhere (although some insights concerning data acquisition and transformation were gained while evaluating the system, and these are discussed in the next chapter). The only transformation issue of concern here is how the system knows the difference among the various types of data in the database, and how it is able to determine the correct numerical value for the system calculations. The distinction among types of data is accomplished by defining a separate data type for each distinct type of data to be handled and storing the name of the data type along with the data. For example,

the prototype uses the simple data type name *textual* for expected textual data, such as ratings of *novice*, *intermediate*, and *expert* (the possible values for the feature *skill level*). The data then determines a numerical value when a rating is needed, using an algorithm specifically defined in the knowledge base for that data type. This allows for unlimited expansion of the various types of data which can be handled by the system.

The Knowledge Structure

Once it was determined that ASSIST would use a set of features and criteria for keywords in the database, the connection between the organization of the data in the database and the knowledge in the knowledge base became a key consideration. It was not expected that the features and criteria would be static elements in this system, but rather they will be expanded, and some of the terminology may change as time goes on. For maximum flexibility, the features and criteria are defined in the knowledge base. The system will access the knowledge base to determine the keywords to be used and how to perform data transformations during data acquisition. It will also access the knowledge base to determine the features and criteria from which the decision maker may choose to make a software selection, and it will use the knowledge base to determine how to translate evaluation data into ratings. Hence, changes to the structure of the features and criteria, and how the system is to deal with them, will not impact the entire ASSIST system, but will be isolated in the knowledge base.

For version 2.0 of the prototype, a representative structure of features and criteria was developed. This is intended as a framework which can be used as a point of departure for ASSIST. It defines an all-encompassing top level of features for specifying absolute software characteristics, and an all-encompassing top level of criteria from which relative software comparisons may be made (see Figure 4.6).

Top Level Features:	Top Level Criteria:
analysis functions	correctness
applied standards	efficiency
associated tool reqmts	expandability
configuration requirements	integrity
contractual matters	interoperability
cost	maintainability
hardware control	reliability
management functions	reusability
numerics	survivability
options	transportability
security issues	usability
source code sizing	vendor support
timing requirements	verifiability
transformation functions	
user profile	

Figure 4.6 - Top level structure of characteristics for ASSIST prototype

Appendix H specifies lower level details for each of the top level features. These details are representative of the types of absolute software characteristics which may be specified in the software selection process. The detail lists in the ASSIST prototype (from Appendix H) are by no means exhaustive. The specification of absolute features can become very detailed and complicated. Of

those listed in Appendix H, some are applicable to any type of software and others are applicable to compilation systems only. However, it is expected that any detail features which would be added to ASSIST would fit under one of the top level feature categories already defined. Multiple levels of details may also be defined.

In similar fashion to Appendix H, Appendix I specifies lower level details for each of the top level criteria. These top level relative software characteristics are often referred to as *quality factors*, and the details from which they are derived are sometimes called *metrics*. The organization of Appendix I is a composite derived from several sources. It has a structure very similar to that of Appendix H, showing the details which determine each of the top level criteria [14], [24], [50], [127]. This is a more agreed-upon structure than that developed for the features, but it is also not intended to be exhaustive. As with the top level of features, the top level of the criteria is expected to be stable, and any extensions or modifications to detail criteria would fit under one of the top level criteria already defined. Multiple levels of criteria may also be defined.

With the definition of the structure of the features and criteria for the ASSIST prototype, it became clear that the terms used for these various characteristics are often defined somewhat differently. Hence, it was necessary to define the terminology used for the features and criteria. Appendix J is a glossary of all terms used in the prototype. The first definition given is the one used in the

prototype, but other commonly used definitions are also included for purposes of clarification.

Now that we have examined the structure of the knowledge base, it is time to look at the system decision making process in more detail. This process structures both the collection of input data from the decision maker and the presentation of output data to the decision maker (in the form of recommendations). In doing so, it relies on the knowledge base for providing the features and criteria to be used, and for providing the mechanisms by which they are used.

Getting Input from the Decision Maker

The decision maker must specify various required and desired characteristics of the software to be selected. The most important issues the design must consider in this area are that the specification must be relatively easy to accomplish and should provide a good match to the user's mental processes. At the same time it must also be complete.

Ease of Accomplishment

How easy a task is to accomplish is relative to one's experience. When looking at the design issues in this area, the idea was to consider the various ways programs usually handle interaction, then try to pick the easiest of these methods from the viewpoint of the user. The following issues were addressed:

1. *Let the user choose from a list rather than requiring a lot of typing.* This is the basic principle of menu-driven systems. Although it can make a user feel restricted if it is not handled well, it has several obvious advantages. Decision makers cannot be expected to be expert typists, and most people will usually feel more comfortable if the use of the keyboard is kept to a minimum. Furthermore, if input is typed, it is prone to error. Choosing from a list avoids all of the frustrations that go along with making sure a response is both spelled correctly and understood by the system [137].

In ASSIST, the lists to be used are provided in the knowledge base. The decision logic system gets the appropriate list of available choices from the knowledge base, and the user makes choices from the list. This technique is simple and effective, both for choices which are mutually exclusive (such as when choosing the type of software) and for lists which permit multiple choices (such as when choosing the important evaluation criteria).

2. *Provide defaults for responses where possible.* For responses which require typed input, if common and reasonable default responses are provided, the user will usually feel that the task is easier. In the case where the default is the user's choice, typing is not even necessary. When the default is not the desired choice, the default gives the user an example of the kind of information which must be entered.

In the case of ASSIST, defaults are provided for all of the numbers used to specify features (such as maximum cost) and for all parameters used in the system calculations. Just as significantly, default suggestions are also provided for the top level features and criteria which usually are important for the given application area. The suggestions used for the prototype are listed in Appendix K. These suggestions can help the user just by the fact that they are the features or criteria which are usually of most concern when dealing with the given application area, but the user may add to or replace members of either list with any of the other top level features or criteria. This can also plant a feeling of confidence in the decision maker that something important has not been overlooked.

3. *Present effectively.* An interactive program is not very useful unless it can present itself to the user in a manner which is easy to understand. In this regard, it is important that the user not be faced with information overload, yet at the same time the system should not appear to be restrictive. To cope with this, a system can present information in meaningful chunks, where each chunk usually contains a single concept or idea, and the chunks are presented in a sequence meaningful to the user [15], [32], [137], [163].

This type of thinking brings us into the realm of hypertext and hypermedia, looking at presentation sequences which are not necessarily static nor linear in nature. Much of the appeal of the hypertext concept comes from the fact that the user does not need to feel restricted by having information presented in only one possible

sequence. However, there are potential dangers when restrictions are lifted. One can get lost while navigating through non-linear information unless the organization of the information makes sense and the user can be reminded of that organization when necessary [9], [28], [32], [38], [73], [128], [140].

In ASSIST, each chunk of information presented to the user is in a separate window. Although these windows are presented in a sequence which is basically linear, the user is not restricted to keeping with this linear presentation. The areas of user specification which can become more involved are also not presented linearly. If the decision maker chooses to specify any features or criteria in detail, this specification is an "aside" from the normal linear sequence. It is a path not taken unless the user chooses to do so, and then it is up to the user to determine how involved and detailed the process will be. The prototype deals with possible problems of the user feeling lost during this process. The mechanism provided is discussed in the next subtopic on support functions.

4. *Provide useful support functions.* Support functions are those which do not specifically address the purpose of the software, but rather they provide more general capabilities which make the software usable. These functions are very important to the impression the user gets about how easy and pleasant the software is to use.

ASSIST defines six support functions, and all of these have been implemented in the prototype. Each provides a capability

which the decision maker may wish to use at any time during the program execution, so all are made available for use at all times. Although it seems clear that this is a desirable feature for any software, support functions can be made available in many ways, and some of these ways are certainly more effective than others. For example, in many systems such functions can be accessed only through the use of obscure key combinations. This is often acceptable to expert users of a system who presumably use the software often enough to be able to remember the key combinations. However, it is not desirable for a system such as ASSIST which would normally get infrequent use by a decision maker.

The prototype takes advantage of the fact that Macintosh applications typically use icons for a visual representation of objects, and it uses icons as HyperCard buttons to represent each of the support functions. Each icon is a picture which is somehow suggestive of the function of the button it represents (see Figure 4.7). The icon buttons are clearly visible on each card, yet they do not take up much space. This leaves most of the card available for presenting a specific chunk of information to the user. With this setup the decision maker always has an unobtrusive visual reminder of the functions available, and each is easily accessed with one mouse click.

Research also indicates that the user needs visual cues which are invariant throughout the chunks of information presented [9], [15], [137]. The prototype accomplishes this by providing the same

information layout on all cards, and this includes a particular area which is always used for the support buttons.

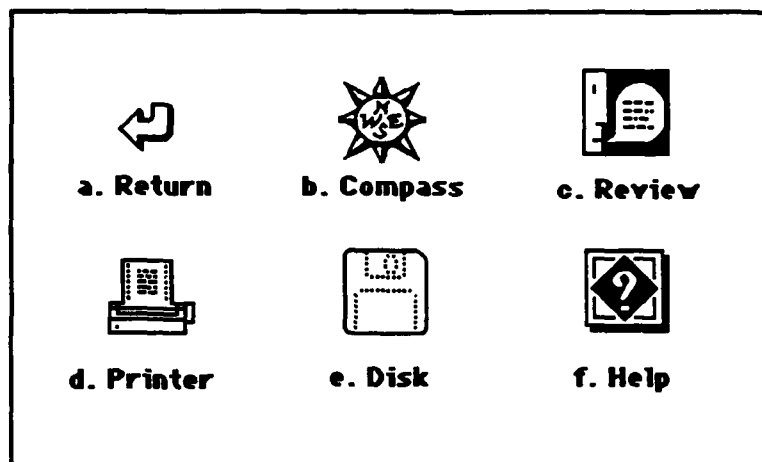


Figure 4.7 - Icons used for the support function buttons

Color can also be used to highlight the icons of the support buttons to emphasize that they may be important to the user. Otherwise, they are more likely to become just a part of the card background which is ignored. Color is not currently a part of the prototype, but future versions will probably use it on the support buttons for emphasis. Any additional color would have to be considered very carefully because it could negate the effect of the color on the buttons. Color can be easily overused, and then it becomes much less helpful [91], [128], [137], [152].

The following is a detailed description of each of the six support functions:

a. Backtracking (see module 1.4.6 in Appendix G)

Backtracking is a very common activity in working with a system providing chunks of information. After looking at a new chunk of information, it is often useful to go back to look at the previous chunk. Since the activity is so common, it makes sense to provide a simple function to accomplish it [9].

This function is so common that the "return arrow" icon button with an arrow pointing down and then to the left (see Figure 4.7a) is already defined by HyperCard. Its function is simply to pop the internal stack. This stack pushes each card which has been the current card as soon as a new card becomes the current card. Popping the stack makes the popped card become the current card again.

This icon button was adapted for the ASSIST prototype (and called simply the "return" icon) both to maintain consistency with other HyperCard applications and because it represents the idea of return as well as any other visual cue. However, to prevent possible confusion, one check was added for the prototype button which comes into play if the user tries to pop an empty stack. If the return button defined by HyperCard tries to pop an empty stack, the current card will become the HyperCard "Home" card. However, the ASSIST user is not assumed to know anything about HyperCard, so the sight of this system "Home" card on the screen could be very confusing. To prevent this problem, if the return button defined by the prototype tries to pop an empty stack, nothing will happen. The current card will just not change. This may be a little disconcerting

to the user, but it is better than being confused by ending up completely out of the ASSIST stack.

b. Graphical browser (see module 1.4.5 in Appendix G)

Browsers can be implemented in many ways, but the main idea is to show a graphic which relates to the organization of the information the user is viewing. The browser indicates the current activity in some manner, usually with a highlight. This lets the user see the relationships between the current activity and others which are associated with it, and it usually prevents the user from feeling lost [38], [73], [86], [169].

The graphical browser in the ASSIST prototype is activated by clicking on the "compass" button (to suggest a means for determining the current location) in Figure 4.7b. It causes the current window to become a window which contains labelled boxes, representing the main activities of the program (see Figure 4.8). Lines connect some of the boxes to indicate the default ordering of the specification activities. The box representing the current activity is highlighted ("chosen features" in Figure 4.8). While viewing the browser, a user may decide to return to the previous window (by clicking on the return button), or a click on any of the activity boxes will go to that activity. Many of the activities are accomplished using multiple windows, and in this case the browser always goes to the first window of the activity.

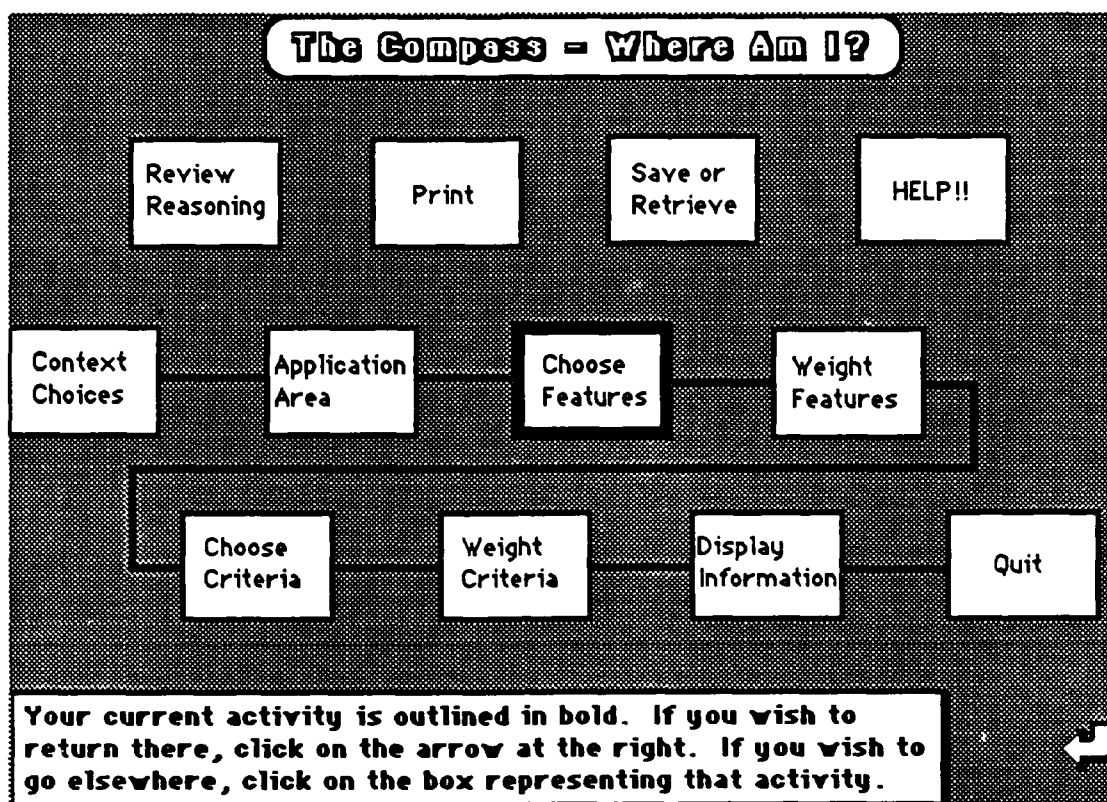


Figure 4.8 - The graphical browser

c. Review (see module 1.4.4 in Appendix G)

The user's ability to review the reasoning process at any time is an important part of an expert system [105], [159]. This idea is also important for the ASSIST design. However, with ASSIST the reasoning comes into play only at the very end of the program processing, when recommendations are made, and the various levels of detail available at that time will take care of explaining as much of the reasoning as the decision maker is interested in. The important thing for the user to be able to review at any time during program execution is the set of specifications which have been made. This is what the "review" button (see Figure 4.7c) activates in the prototype.

The icon for the button shows a cartoon-type image of someone talking, suggesting that this button will help one to get information.

d. Printing (see module 1.4.3 in Appendix G)

The ability to send information to a printer for later review is important to most people. The ASSIST prototype permits the decision maker to send any card image to the printer. The formal design permits informative reports to be printed also. The button used to activate this activity is the "printer" button (see Figure 4.7d), and its icon is the image of a printer.

e. Save and retrieve specifications (see module 1.4.2 in Appendix G)

Although it is possible that a decision maker will execute this program only once to get recommendations and enough information for making a decision, it is also very likely that ASSIST will be executed more than once by the same user. For example, the decision maker may need to make decisions concerning several different types of software. Another likely possibility is that the decision maker will want to perform a type of sensitivity analysis to determine how much the recommendations will change with the change of just one of the chosen specifications. It also could be that a detailed examination of ASSIST's recommendations may indicate that an evaluation of a particular characteristic of interest in a candidate implementation is not in ASSIST's database. In this case, the decision maker may engage some technical staff members to perform that

particular evaluation. It would be desirable then to have that data added to the ASSIST database and then run the program again.

In instances such as these, it is very likely that the decision maker would want to use the same, or almost the same, specifications for the software to be selected as in the previous run. For such occasions, the save and retrieve activity is provided in ASSIST. The user's specifications may be saved to a file at any time during program execution, no matter how complete or incomplete they are. When they are retrieved, the saved specifications will override any which may already have been chosen by the user. If the specifications were incomplete when saved, then all of the choices which were not made will still be incomplete when retrieved. In other words, even if specifications were made before the retrieval, they will be wiped out.

In the prototype, the specifications will always be saved to and retrieved from a particular file name. However, in the full design, the user is able to choose file names. Since even naive computer users are familiar with the concept of saving information to a disk, the "disk" icon (see Figure 4.7e) is used for the button to access this activity.

f. On-line help (see module 1.4.1 in Appendix G)

On-line help facilities have become so commonplace as to be expected in any good piece of software, and this is as it should be. Unfortunately, the quality of on-line help has not always been very good, often consisting of very curt, stiff textual comments. However,

with large memories, graphics capabilities, and hypertext software available today, there is no excuse for continuing some of the poor practices of old [128].

ASSIST provides on-line help facilities, and the prototype takes advantage of the HyperCard implementation, using its hypertext capabilities. This permits such features as going directly to the help information related to the current card, providing as much information as may be helpful to the user, and giving the user the ability to navigate freely and easily through all of the help information available for ASSIST in whatever order is desired. The icon used for the "help" button (see Figure 4.7f) is a question mark. This is a symbol very commonly used for help, so it was a natural choice.

Completeness

ASSIST was designed for use by every type of decision maker, from those with backgrounds which are highly technical to those who have largely managerial backgrounds. Many of these decision makers will not be concerned about specifying the software to be selected in great detail, rather they will want to specify general characteristics of importance. Others will demand the flexibility to be able to specify almost any detailed characteristic about the software when desirable. ASSIST accommodates all types of decision makers by making various levels of specification detail possible.

If the basic specification process is not straightforward enough, the non-technically oriented decision maker will not use ASSIST. On

the other hand, if the capability for detail is not present, the decision maker with specific technical concerns will not use it. Most systems do not attempt to target both extremes of users, and they concentrate on the concerns of only one of the extremes. However, ASSIST targets decision makers of all backgrounds, and it is specifically designed to make any type of user feel comfortable.

This is a very ambitious design goal, and it is not easily accomplished, but the very concepts which are used in hypertext make this possible to achieve. The idea is to present the decision maker with only the minimum required amount of specification as the default path through the program. This keeps the non-technical user from having to wade through undesirable levels of technical details. However, the technically-oriented user will have the detailed levels of specification readily available at the click of the mouse. The paths to the details will be asides from the default path through ASSIST, but it is imperative that they be easily accessible, and that the user can get back to the default path just as easily when desired.

The following discussion looks at each of the steps in the path through the specification process in detail. With each of these steps, any asides which may be visited for more detailed specification are also examined.

1. *Define the type of software to be selected* (see module 1.2.2 in Appendix G). The prototype refers to this as the specification of the context, and it is required so ASSIST may determine which

software implementations in its database are candidates for consideration. This information is also required for ASSIST to know how to deal with the type of software to be selected by the decision maker. For example, if the user is interested in a set of project management tools, ASSIST will not only look at evaluations of such tool sets, but it will also look at evaluations of each of the tools which make up the tool set.

As discussed in the previous chapter, the user may specify software which is an individual tool, a tool set, support for a life cycle activity, or a whole APSE. After some use by decision makers, other categories of software collections may also be added to these, but this will require no design change in ASSIST. Additions need only be made to the knowledge base. The knowledge base requires an entry in an index for identifying the components of the new choice, and it requires an entry for the new choice with the knowledge of how to deal with its evaluations.

2. Define the application area of the software to be selected (see module 1.2.3 in Appendix G). For this activity the user simply chooses one out of a list of possible application areas. Any number of application areas may be defined by ASSIST, simply by indicating each in the knowledge base, along with an accompanying list of suggested features and criteria of importance. However, the usefulness of this part of the specification is probably best served by a relatively small number of choices. The main differences in the features and criteria which are usually important for the application

come when making comparisons between real time and non-real time software. The list of choices could be expanded a little to differentiate a few more broad categories, but beyond that it gets more confusing rather than helpful.

3. *Define the important features of the software to be selected* (see module 1.2.4 in Appendix G). This is the place where the decision maker may specify any particular requirement which the software must satisfy, or any other absolute characteristic of the software. For example, an APSE may have to run on a particular hardware/operating system configuration, or its cost may not exceed a particular dollar figure.

The decision maker is first presented with a list of suggested top level features. Any number of the suggested features may be chosen. In addition, the user may add features at any time from the list of remaining top level features. The only restriction is that the list of chosen important features may not exceed a system-defined maximum. If the list already contains the maximum number of items, adding a new feature will require the user to discard one of the old ones.

When top level features have been chosen, lower level features usually must be chosen to complete the specification. When lower level details are necessary, choices are presented in a separate window in the same manner as the top level features. When specific values are required for a choice (such as the highest cost allowed or the skill level required), the user enters the value and it is then

displayed along with the feature as a reference for the user. ASSIST does not restrict the number of levels which may be defined in specifying the features. However, the prototype only uses one level of detail.

In the formal ASSIST design, the system must set a maximum length for a list, and this maximum may not exceed 9 items. Research suggests that a human being can cope well with 7 plus or minus 2 concepts at one time, but when this principle is violated and the number of concepts become too great, the person's effectiveness diminishes rapidly [23], [111], [137]. Multiple levels of detail are used in ASSIST to permit all important issues to be addressed while still not violating this principle. The prototype sets the maximum list length at 8 since that is a size which fits comfortably in a window without causing clutter.

It could still be argued that a decision maker may want to choose every feature on the top level list, and this would exceed the system maximum. Although this seems unlikely, it is possible. However, in this case the user will be asked to supply the *most* important of the important features. It remains to be seen if this provides the decision maker with a feeling of being restricted too much.

4. *Assign relative weights to the chosen features* (see module 1.2.4 in Appendix G). Once the user has completed the process of choosing the features of importance, default weights are assigned to these features. The decision maker then reviews these defaults and

makes changes where desired. Defaults are presented for both the top level and the detail level features. In the prototype the top level defaults differ depending on the application area, but the detail defaults are always the same. This could be changed to provide even more help for the user by varying the detail defaults as well.

Relative weighting can be represented in many ways, and the theories concerning the process can get rather involved. Alternatives can be rank ordered, they can be assigned a value on an arbitrary scale, or each alternative can account for a fraction of the whole worth of the decision to be made [64], [74] [143], [165], [172]. In the ASSIST prototype, the weights are assigned on an arbitrary scale from 1 to 10. This was used because it is a concept common in every day use, so it can be easily understood by the user, and there is no reason to believe that it would be worse than any other method for assigning values of arbitrary worth based on human judgement. In effect, values are assigned from 0 to 10, where 0 is the weight given to any feature which is not considered important enough to choose.

When choosing the important features of a piece of software, the decision maker sometimes needs to be able to stipulate that certain features are absolutely essential while others are just desirable. This can be accomplished with the weighting system in ASSIST. In the prototype, a weight of 10 indicates that a detail feature is absolutely essential, and an implementation will not even be rated if the feature is not present. Weights of 1 through 9, on the other hand, indicate how desirable a feature is, but absence of that feature will not necessarily make the software unacceptable.

5. Define the important criteria against which the software to be selected must be evaluated (see module 1.2.5 in Appendix G). At this point the type of software to be selected has been identified, and any absolute characteristics which are important have been specified as features. Conceptually, this has narrowed the field down to only the candidate software implementations. Now is the time to specify those important relative characteristics against which these candidates must be evaluated to determine which are "good enough" to be selected for the decision maker's purpose.

The decision maker is presented with a list of top level criteria in much the same manner as the presentation of top level features. The same maximum number applies to the criteria choices for the same reason. However, there are some significant differences in the way criteria are specified.

Criteria are not as specific and detailed as features, and so it is not necessary for lower level details to be specified. Since the decision maker may not wish to deal with details, they are not automatically processed. Rather, the user may choose whether to specify details or not. If the choice is to specify details, each detail list is presented in a separate window in much the same manner as for feature detail specification. If the user chooses not to specify criteria details, the system will automatically use all the detail criteria defined for each chosen top level criterion for which there is data in the database. Again, the prototype implements only one

level of detail, but the complete design does not restrict the number of levels which can be used.

6. *Assign relative weights to the chosen criteria* (see module 1.2.5 in Appendix G). This process is essentially the same as the process for assigning weights to the chosen features. In the prototype, there are only two differences. If the user chose not to specify criteria details, then equal default weights are assigned to all detail criteria used, and they are not changeable. Also a weight of 10 does not have the same meaning for a criterion as for a feature. Because of the relative nature of criteria, it makes no sense to say that a criterion is absolutely essential. Criteria are neither present nor absent, but rather exist along a continuum of "goodness". Hence, a 10 indicates that a good rating with respect to that criterion is critically important, but a poor rating will not necessarily make the software unacceptable.

7. *Set the system parameters* (see module 1.5.1 in Appendix G). Several parameters are also required for the calculations done to produce recommendations for the user. Defaults are automatically provided at system start-up. However, these are important parameters, and the user should be permitted to change them. The first is the determination of the relative overall importance of the set of features and set of criteria chosen, and the second is the rating which will be considered "acceptable" when preparing the presentation of the recommendations to the user.

However, the specification of these parameters produces a small dilemma. If the user does not have the opportunity to change them before the calculations are done, then at least the first set of recommendations will be based on the default values of the parameters. On the other hand, explaining the parameters and how they affect the calculations is a detail which may be confusing and/or of little concern to some users. Such explanations are a part of the second level of detail for presenting recommendations to the user because they help to describe the process used to arrive at the recommendations. It is not expected that all decision makers will wish to see this level of detail.

The dilemma was resolved in the prototype by presenting the parameters to the user via a button in the display window in which recommendations are presented. This means that one calculation will often be done using the default parameters before a user realizes the parameters can be changed. However, the type of user who would get to the recommendations before realizing this possibility would either be a decision maker who is not interested in the details anyway, or else one who wants to run the program once to get a feel for it before reading any documentation. In either case, nothing is lost, and an expert user who is interested in changing the parameters can go to the display window via the browser and make the changes before any calculations are done. By not making the parameter settings a normal part of the default sequence of choosing, ASSIST remains a system that a decision maker can go through in a very simple, straightforward manner. This simplicity was deemed to

be a very important feature to make ASSIST acceptable to all types of decision makers.

Providing Recommendations to the Decision Maker

Once the entire specification process is complete, ASSIST must do a considerable amount of processing and calculating before recommendations can be presented to the user. There are two types of issues which must be addressed. ASSIST must perform calculations which reflect comprehensive and consistent assessments of the candidate implementations (based on the data in the database and the knowledge in the knowledge base), and it must present reasonable recommendations to the decision maker based on these calculations. These issues are addressed in detail below.

The Calculations Involved in Ratings

First we will look at the calculations performed by ASSIST. Since many of the calculations are the same for features and criteria, the first two steps of the discussion just refer to characteristics. The later steps discuss how the separate feature and criteria assessments are combined.

1. *Turn each assessment relating to a specified characteristic into a numerical rating* (see module 2.1.4.5 in Appendix G). As discussed previously, the knowledge base contains the algorithm which determines how to turn assessment data for any particular data type for each characteristic into a numerical rating. The system obtains the data about a characteristic from the database and sends

this, along with the user specification data (for a feature), to the knowledge base. The knowledge base then determines a rating for that particular characteristic of that software product.

2. Deal with multiple, possibly conflicting, assessments of the implementation with respect to a specific characteristic, and arrive at a single rating for the implementation with respect to that characteristic (see module 2.1.4.3 in Appendix G). This is an area of concern in problems using a knowledge base because conflicting information is always a possibility, and it can be handled in many ways. In addition to dealing with conflicting assessments, incomplete assessments should also be considered [56].

In an application like ASSIST, this problem is not difficult to deal with. In other applications, conflicting information indicates that the knowledge base has some "bad" information to work with, and the problem is then to determine which is the bad information so it can be thrown out. In ASSIST, however, the conflicting information is usually a result of various opinions. No opinion is either right or wrong, and no one opinion should be considered better than another.

Thus, the way ASSIST deals with conflicting opinions is to combine them together to arrive at a composite assessment. This is easy to do since the assessments end up with a numerical value. The result is that the combination of opinions of each type will determine the final rating for the characteristic in question. However, the source or the date of an assessment may also make a difference to

the decision maker, so ASSIST makes allowances for this also. When examining the details of the recommendations, the user will have the option to indicate that assessments from certain sources (such as from the software vendor) should be excluded. The user can also indicate that only assessments which have been done since a particular date should be included. In this way a decision maker can eliminate a perceived bias in the assessments, and timeliness of the information used can be assured.

It is possible that in certain cases the conflicting information is not based on opinion. For example, errant data could have been entered into the database. This is not considered a problem for the decision logic of ASSIST, however. It is a problem which must be dealt with by the Knowledge Acquisition Subsystem, which is not specified in detail in this research. For the purposes of this research, all the data being processed by the knowledge base must be considered to be valid.

Missing data is a different problem. ASSIST simply rates a characteristic with a 0 if no data exists for it in the database. However, information about which data was missing can be important to the decision maker, so the design requires that such information be made available to the user (see module 1.5.2 in Appendix G). This is discussed further below under "Presenting the Results".

3. Combine all feature ratings and all criteria ratings to arrive at an overall rating for each implementation (see module 2.1.4.4 in

Appendix G). In calculating the combined overall ratings for the features and criteria, the issue of their independence or dependence had to be dealt with. The reader will recall that in the last chapter (in the section "The Calculations") we determined the validity of using a linear additive function to combine ratings as long as the ratings represented independent characteristics. Hence, this method was used for independent combinations. The problem, then, was to determine how to treat ratings for dependent characteristics, as well as how to determine which characteristics were dependent and which were independent.

As it turned out, once the method for choosing features was worked out, this was not a problem for feature calculations. The top level features are all clearly independent. With the levels of detail used for the feature choices, the choices available for the detail of any particular feature will not only be the features which describe it in more detail, but they will also be ones on which the higher level feature is dependent. In other words, the hierarchy used in the process of choosing features is also a dependency hierarchy.

This structure of features could be said to form a decision tree, where each feature is a node of the tree and the root is the composite level above all of the highest level features. The validity of the resulting decision (or rating in this case) is dependent upon whether the relationships at each node of the tree are correct. Each node is dependent upon its lower level nodes, but the lower level nodes for any particular node are independent of each other [64], [131], [143].

The features at any particular level are independent of each other, and the calculations (using the linear additive function) for each level are done separately. The algorithm used to produce the overall feature rating for a candidate implementation (software product) in the prototype is given in Figure 4.9. It consists of an outer loop for calculating the ratings for the chosen top level features, and an inner loop for calculating the ratings for the chosen detail features. The procedure for each loop is similar.

```

set the top feature ratings to 0
set the top feature weights to 0

loop for each top level feature chosen
  set the detail feature ratings to 0
  set the detail feature weights to 0

  loop for each detail feature chosen
    retrieve rating
    add feature weight * detail feature rating to detail feature ratings
    add feature weight to detail feature weights
  end loop

  top feature rating = detail feature ratings / detail feature weights
  add feature weight * top feature rating to top feature ratings
  add feature weight to top feature weights
end loop

overall feature rating = top feature ratings / top feature weights

```

Figure 4.9 - Algorithm for calculating an overall feature rating

For the inside loop, the rating for each detail of a particular top level feature is multiplied by its weight, and all of the weighted ratings are totalled and divided by the sum of the weights. This

results in the top level feature rating. In the outside loop, the rating for each top level feature is multiplied by its weight, and all of the weighted ratings are totalled and divided by the sum of the top level weights. This results in a weighted overall feature rating.

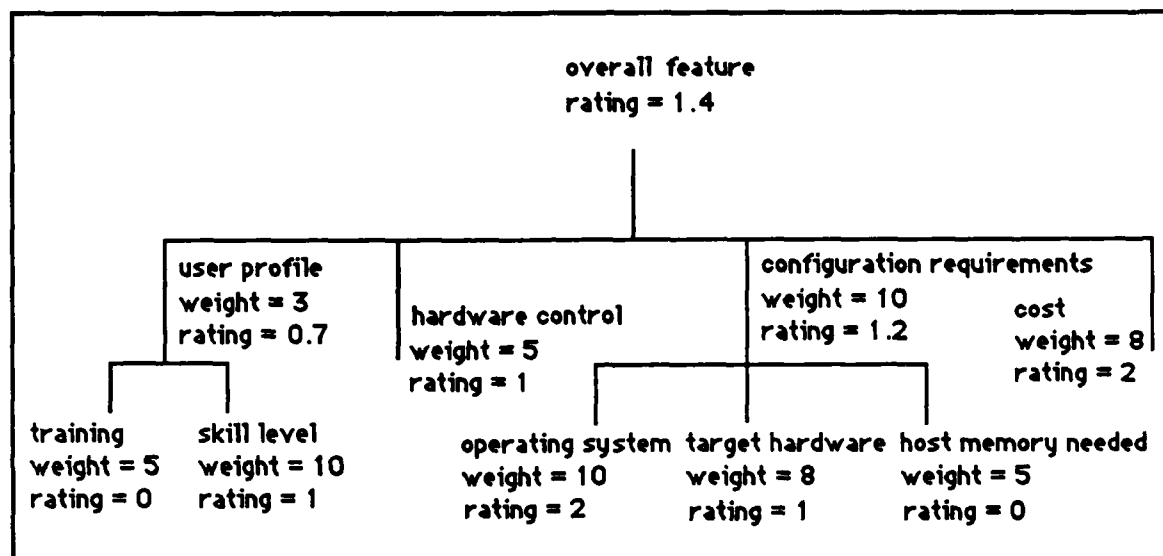


Figure 4.10 - Calculations for an overall feature rating

An example of these calculations is shown in Figure 4.10. In this example, the top level features cost and hardware control do not have detail levels of features specified, so their ratings are determined by the knowledge base, based on the data in the database for these features. On the other hand, configuration requirements and user profile have detail specifications, so the ratings for their detail features are determined from the data in the database. Each top level feature rating is calculated as the weighted average of its details, as described in the algorithm in Figure 4.9.

Similarly, the overall feature rating is calculated as the weighted average of all the top level features.

This algorithm works for the prototype, but if more than one level of detail were used in ASSIST, a modification would be necessary. The modification would require a recursive call on the detail feature loop for each additional level of detail (a looping structure equivalent to the recursion could also be used).

The criteria calculations should not necessarily be done using the same algorithm as that used in the feature calculations. Some of the top level criteria are not independent because they depend on some of the same detail criteria. For example, reliability and maintainability each have consistency, modularity, and simplicity as some of their detail criteria [14], [24], [50], [127]. Hence, strictly speaking, criteria rating calculations may not be valid if done using the same linear additive function used for feature calculations.

However, the literature on decision theory provides no basis for using any other particular function in this situation either. Usually independence is assumed, and even when it is not, discussions of other functions do not give clear-cut conditions for using them. Furthermore, most authors emphasize the difficulty in determining independence or dependence as well as the subjectivity of the decision process. The numerical nature of any of the decision functions is misleading because they give the impression of precision when the numbers are based on subjective judgements [64], [74], [165], [172].

For all of these reasons, the criteria calculations in ASSIST are done in the same manner as the feature calculations. It is possible that experience with ASSIST will dictate a change in the function used for the calculations, but in that case, there will be empirical evidence to help determine what the new function should be. In the meantime, the linear additive calculations should at least be close, and this author suspects that close is as good as can be achieved anyway. Later changes to the functions used for criteria calculations, if deemed appropriate, would require a modification only to the knowledge base.

Once an overall rating has been calculated for the features and another for the criteria, these ratings must be combined somehow into one final rating for the software implementation (see module 1.5.4 in Appendix G). The default method is simple, just averaging them. However, the user may or may not want them to carry equal weight. The weights for these ratings are parameters in the system which the user may change. In the prototype, the weights are given on the same 1 to 10 scale used for the other system weights to avoid confusion. Figure 4.11 illustrates the algorithm used to calculate the final rating, which is a weighted average of the overall feature and criteria ratings.

The prototype system saves the overall feature and overall criteria ratings as well as the final overall rating for each implementation. These will be reported to the decision maker if details of the calculations are requested. The prototype will also describe the calculation process, but it does not give any additional

figures. In a more complete implementation of ASSIST, the results of all the calculations would be kept for even more detailed reporting.

weighted feature rating = overall feature weight * top feature ratings
weighted criteria rating = overall criteria weight * top criteria ratings
overall rating sum = weighted feature rating + weighted criteria rating
overall weight sum = overall feature weight + overall criteria weight
final rating = overall rating sum / overall weight sum

Figure 4.11 - Algorithm for calculating an overall rating for one product

The results of the calculations which are kept are rounded to one digit after the decimal point. Although the literature will often show calculated results of apparently much more significance, in actuality even reporting a significance of two digits is questionable. The one digit ratings are subjective in the first place, as discussed above. After these are multiplied by weights of one significant digit, it is impossible to obtain a result which is truly meaningful beyond one significant digit [53], [138], [154]. However, an extra digit should be kept in the calculating process, so this extra digit is also shown to the decision maker when details of the calculations are requested.

Presenting the Results

A number of important issues must be addressed with respect to the presentation of information to the decision maker. Some of these, such as the timeliness of the information, have already been addressed. Others require a closer look.

It is the presentation which will assist the user in making a software selection decision. No matter how good the information is, if it is not presented in a form which is easy to understand and use, ASSIST will not accomplish its purpose. The recommendations and other information must be presented in a form which is, at the same time, both concise and complete.

ASSIST was designed for use by every type of decision maker, from backgrounds which are highly technical to those which are largely managerial. For many of these decision makers, the lower levels of technical detail about various software evaluations will not be a concern. For others, certain technical details will be a critical part of making a good decision. ASSIST accommodates all types of decision makers by making various levels of detail available and giving the user the option of how much detail to look at (see module 1.5.4 in Appendix G). The highest level of detail takes care of providing a concise, though comprehensive, presentation of recommendations. It is up to the lower levels to provide complete information in as much detail as the user desires.

Many of the aspects of the highest level of detail, which presents the recommended software implementations to the user, have already been discussed. This level purposely does not give numerical ratings for the implementations because ASSIST does not want to convey a message to the user that the methods used in arriving at the ratings are objective or precise when just the opposite is true. At best the ratings give indications of trends and clear distinctions in quality.

Keeping this in mind, the highest level of detail lists the software implementations of the chosen type in one of three categories. The first list names those implementations which are rated "acceptable", based on the user's specification of the system parameter for the level of acceptance. The implementations are listed in order of ratings, from highest to lowest. Along with the identity of an implementation is an indication of how many different evaluations were used from the database to arrive at this recommendation. This is given to help the decision maker gain a better perspective on the comparisons among the implementations.

The second list gives the same information, but it names all of the implementations which came out with "unacceptable" ratings. These implementations had the required features, but they just did not compare favorably with those on the "acceptable" list.

The last list names those implementations which were not even considered because they did not have all of the required features stipulated by the user. This is another bit of information aimed at helping the decision maker get a perspective on the recommendations given. The combination of the three lists lets the user know exactly what implementations are included in the system database.

If this information is not enough for the decision maker, lower levels of detail may be desirable. The first type of detail describes the calculations which determine the ratings of the software implementations and lets the decision maker see the results of the calculations. The prototype goes no farther than describing the

algorithm used in the calculations and showing the overall feature and criteria ratings for each implementation as well as the overall rating for each. However, the design allows for additional details on any part of the rating process. This could be expanded to show all the details of the calculations as well as the details of how the data from the database is converted into individual ratings. At the lowest level of detail, the decision maker could actually view pieces of the data in the database that are of interest, and pointers could be given to the evaluation reports which produced the data.

The important thing is that the presentation is made at various levels of detail so that information overload does not occur. The user should also be able to feel a sense of closure at any point so it does not seem necessary to ask for more detail unless there is truly more detail of interest. In other words, each detail window should be a complete picture at some level of abstraction rather than just a part of a continued description.

Another type of detail which should be available to the decision maker is additional perspective information about the data missing from the database. This information is available in ASSIST, and it is implemented in the prototype (see module 1.5.2 in Appendix G). When the user chooses a particular criterion against which to evaluate an implementation, and one or more of the candidate implementations have not been evaluated with respect to that criterion, those implementations are not necessarily getting a fair comparison with other implementations. Given details on missing information, the decision maker knows which features and

criteria require assessment data. If this appears to be a significant problem, the decision on the software selection could be postponed until some additional evaluations have been performed.

A significant result of a design which makes information available at various levels of detail is that the system is extensible. Even more details can be added to the system without changing the design. Experience with the prototype may well bring out requirements for some additional types of information which have not been anticipated.

An integral part of the completion of the system design was that a prototype with a reasonable subset of capabilities was also completed. This permits the design concepts to be used and evaluated, and modifications can then be made based on any problems which are identified in the working model. This will be an ongoing process.

However, what remained for this current research was an evaluation of the work that had been done. This evaluation looked at the formal design to determine if it was technically complete and appropriate in every way. It also looked at the ASSIST prototype to evaluate the usability of the design through the execution of further scenarios. The scenarios specifically looked at how the prototype deals with the assessment of Ada compilation systems. The next chapter details this evaluation process.

5. Evaluation

In this chapter, the full ASSIST design is evaluated. The prototype is used as an example illustrating many of the important aspects of the design. A number of evaluation schemes could have been used in this evaluation. However, since this research has developed an all-encompassing framework of features and criteria for organizing software evaluations, nothing seemed more appropriate than to use this framework for discussing the evaluation of ASSIST.

The first part of this chapter evaluates the ASSIST design, using as many of the ASSIST features and criteria (see Appendices H, I, and J) as are appropriate. Because many issues are involved in evaluating software usability and because many aspects of usability can be more effectively evaluated from a working program, the second part of the chapter provides additional elaboration on the usability of ASSIST through the analysis of the execution of two scenarios on the prototype. This provides a basis for the discussion of some of the types of evaluation information necessary for evaluating compilation systems. Additionally, it provides a forum for discussing the insights which grew out of the effort to populate the database of the prototype to prepare it for execution. This evaluation of the usability of the prototype also provides a measure for evaluating the potential usability of a full implementation of the ASSIST design. The chapter concludes with a final overall analysis of the ASSIST design, including user evaluations of the prototype.

Evaluating the ASSIST Design

To evaluate the design as completely as possible, several issues must be addressed. The ASSIST features and criteria are meant for use in the evaluation of a software product, but ASSIST is not currently a completed product. However, the design is a blueprint for the software product to be developed. As such, many aspects of the ASSIST features and criteria pertain to its design. For a more complete evaluation, it will also be useful to discuss the features and criteria as they apply to the prototype implementation as well as to other potential implementations of the ASSIST design. In this way, all pertinent aspects of the design will be covered.

Each of the top level features and criteria will be discussed individually. First the features will be analyzed, including those which are required by the design, those which are present in the prototype, and those which will be significant if they become a part of a complete implementation. An analysis of the criteria will follow. They will be discussed in terms of how well they are supported by both the design and by the prototype. In addition, the issues relating to those criteria which will be important in a complete ASSIST implementation will be addressed.

Features

In discussing the features, it is important to remember that these are the absolute characteristics of the software. They determine the existence of function as well as many other aspects of what exists as a part of and in conjunction with a software

implementation. An actual assessment rating cannot be derived from most of this feature information until a decision maker specifies requirements for the desired software.

analysis functions

ASSIST may perform several functions which fall under this category. It does an analysis of the assessment information in its database as a part of its recommendations to the decision maker. Hence, the design allows for such analyses as decision analysis. It also may provide statistical profiling, which could be an important part of analyzing compiler performance, for example.

The ASSIST prototype implementation itself performs a requirements simulation function, which is also an analysis function.

applied standards

Although ASSIST deals with Ada software and it has been designed with the intention of an Ada implementation, it does not apply or adhere to any standards in its prototype implementation. The design has employed the use of IEEE standards in the design process. However, these standards are actually guidelines rather than strict standards.

associated tool requirements

The implementation of ASSIST will determine what tools may be necessary for use in conjunction with it. No particular tool is required in the design. However, the design requires a database, and it is likely that an implementation would take advantage of an

existing database system. The only tool which is currently required for running the prototype is HyperCard because the database is built into the HyperCard stack which is the ASSIST prototype.

configuration requirements

The required configuration will also be determined by the eventual full implementation of ASSIST. The only required peripheral is a printer. The prototype requires a Macintosh which can run HyperCard. This means it must be a Macintosh Plus, an SE, or any of the Macintosh II line. One megabyte of primary memory and two megabytes of disk space are required (which means that a hard disk is required). Of course, a full implementation with a substantial database would require considerably more disk space. The prototype places no restriction on the type of printer which is used.

contractual matters

Contractual matters associated with the distribution of ASSIST will be determined by the implementor. The prototype has been developed as a part of U. S. government funded research. Hence, it may be provided in its current form to any interested government organization without charge (except possibly for materials and postage).

cost

Cost is dealt with separately from contractual matters because only the cost of the actual software is necessarily a part of the

original system acquisition. Other costs associated with making the software a productive part of the user's facility, such as training and installation costs, are also dealt with in this category.

The cost of acquisition is specifically determined by the vendor. Other costs are influenced by the way the system is implemented, but they will vary greatly, and they are much more difficult to specify exactly.

For the prototype, no cost is associated with the actual software. Since the decision maker is expected to know nothing of the mechanics of using the system, beyond how to turn it on, start the application (and these could even be done for the user), and use a mouse, no training costs are expected in this area. However, it should be noted that the user must have some knowledge of software evaluation technology in order to use the system intelligently. This may or may not require training.

For installation, the only expected expense might be the purchase of HyperCard if it is not already on the system. However, HyperCard has come bundled with all appropriate Macintosh systems since its initial release about two years ago, so it is likely to be already in place.

hardware control

The ASSIST design specifies no direct hardware control.

management functions

The management function provided by ASSIST is its primary overall function, decision support.

numerics

The numerics used in ASSIST are not complicated, so it would be expected that any system would be able to handle them. The most important consideration is that all calculations use only two digit precision because additional precision makes no sense with the imprecise numbers used in this type of decision analysis.

options

The decision maker using ASSIST has the option to specify features and criteria to the desired level of detail and to view recommendations at the desired level of detail. In the prototype, the user must specify one level of detail for most features, but criteria details may be bypassed entirely. The detail recommendations may also be bypassed. However, none of these options requires a special external set-up. They are all an integrated part of program execution.

security issues

The ASSIST system itself is unclassified. However, it is conceivable that classified data could be used in its database, and in that case the system would take on the classification of the data.

The prototype does not deal with security issues. However, in a full implementation, these issues could become critical to a software selection decision.

source code sizing

The important issue here is that the ASSIST design does not restrict the size of the system database or the knowledge base. The only limits will be the practical limits on the use of disk space on the system on which ASSIST will run.

timing requirements

The ASSIST requirements have specified certain levels of system response times. In the case where possibly long delays are expected, when there is highly concentrated activity in the database, requirements specify that the user be informed of progress at least every 30 seconds. For even short delays, the system must always give the user feedback indicating that something is happening.

The prototype adequately addresses the system timing requirements, although a prototype cannot be expected to necessarily meet all such requirements. It is concerned with proof of concept rather than performance. The prototype provides the required user feedback during processing. During all delays, the cursor turns into a spinning ball, giving a clear visual indication that the system is working. For expected long delays, a message to that effect is always displayed, and when the system is processing all the

data in the database to arrive at recommendations, progress messages are displayed.

transformation functions

The ASSIST design specifies two types of transformation functions. When data acquisition takes place, the raw data will be transformed into a form appropriate for the ASSIST database (this is discussed in more detail in the next section on executing scenarios on the prototype). When the system is processing recommendations, another transformation takes place. The data in the database for each feature or criterion of each software product is transformed into a numerical rating for the feature or criterion. This rating is then used in the system calculations which result in the final recommendations.

user profile

The decision maker is expected to require little or no training in the mechanics of how to run ASSIST, and the required skill level for using ASSIST is that of a novice. However, it should be noted that the more the decision maker understands of evaluation technology, the more effectively this system can be used.

ASSIST's database and knowledge base are set up by another user who is termed the system manager. The system manager function could be performed by the ASSIST implementor or by someone in the using organization. In either case, the skill level required will depend to some extent on the design of the Knowledge

Acquisition Subsystem (which has not been completed in this research), but the required skills will certainly include a considerable knowledge of evaluation technology.

Criteria

In discussing the criteria, it is important to remember that these are the relative characteristics of the software. Each criterion exists to some degree in every software product, and its rating is a determination of the "goodness" of the software with respect to the criterion. A rating may fall anywhere on a continuum from very good to very poor, and it is much more subjective in nature than a rating for a feature.

The ratings for these top level criteria are derived from the ratings for the lower level details which are specified as important by the decision maker. Details which are not important to the decision maker are not considered in the rating. Hence, the criteria ratings will be different for different situations, as well as being different for different raters.

In the discussion below, each of the detail criteria which has been identified as contributing to a top level criterion is considered as a part of the evaluation of ASSIST with respect to the top level criterion.

correctness

Correctness is determined by the detail criteria completeness, consistency, and traceability. The ASSIST design is not complete in

that it is only a high level design. It has also not undergone rigorous reviews. Hence, although every attempt has been made to arrive at complete requirements and a complete high level design, they will undoubtedly become more complete as the detail design work is accomplished. The use of a prototype has helped this effort tremendously up to this point, and it will continue helping as long as its development continues along with the design process.

Much attention has been given to consistency in ASSIST throughout the design process. The user interface is the place where consistency, or the lack thereof, is very visible, and the design addresses the issues which have been identified here. Consistency is also important in the system calculations, and this has been analyzed carefully as well. The prototype illustrates the consistent "look and feel" of a good user interface, as well as the use of consistent calculations.

Traceability has been built into the ASSIST design. All design modules (see Appendix G) reference the requirements as well as the tests which apply to them. The Requirements Cross-Reference Matrix in Appendix F also provides this traceability from requirements to both design elements and tests. Finally, the Test Descriptions in Appendix E provide traceability from each test back to both the requirements and the design modules.

efficiency

System efficiency is determined by the detail criteria communication effectiveness, processing effectiveness, and storage

effectiveness. The ASSIST design requires no external communication resources aside from those required for the expected communication with a hard disk and a printer, and the effectiveness of those resources are system dependent. Processing and storage effectiveness cannot be assessed until the design is implemented. In the prototype implementation, these are by no means optimal. Some improvement in processing effectiveness has been accomplished through the use of a HyperTalk compiler utility (HyperTalk is normally interpreted), but a production quality system would be expected to be much more efficient. The use of a database system which is currently available for use on a Macintosh could probably improve both the processing effectiveness and the storage effectiveness of the prototype.

expandability

Expandability is determined by the detail criteria augmentability, generality, modularity, self documentation, and simplicity. ASSIST has been designed to be augmented. The structure of the features and criteria used for specifying the software to be selected, as well as the algorithms used in conjunction with these features and criteria, are confined to the system knowledge base. This permits the straightforward addition of any number of new feature and criteria categories and details, as well as any number of modifications to those which already exist. Also, because any additions or changes to the system ratings and calculations are confined in the knowledge base, these can be accomplished with

relative ease as well. The object-oriented design has also provided a structure in which the system objects and their associated functions could be isolated. This has resulted in a low degree of coupling among the system modules.

The ASSIST structure is intended to be valid for software evaluations in general. Although the terminology and some of the features are somewhat specific for Ada software, there is nothing in the basic design which is not also appropriate for evaluating any other type of software as well.

Modularity is an important part of the ASSIST design. Not only have functions and algorithms been modularized using the common concepts of subprograms, but objects have been encapsulated using the Ada package concept. Within the limits of HyperCard, modularity has been effected as well as possible in the prototype.

Self documentation applies to the documentation in source code, so this cannot apply to the ASSIST design. For the prototype, each source code module has been documented with a description of what it does, an explanation of each argument used, if any, and an explanation of what is returned, if anything. In addition, comments are strategically placed at the beginning of the major blocks of code within each module, and other comments have also been added for clarification wherever they seemed to be helpful.

The simplicity (or lack of complexity) of a system can be discussed in terms of such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the type of data structures. The high level

design of ASSIST does not get to very much detail in these areas, but the basic structure of the design has been kept relatively simple and straightforward. The structure of the subsystems and the interfaces for each object structure have been defined with simplicity in mind. In the prototype, interfaces have been kept relatively simple, with most accesses requiring no more than one argument. The degree of nesting does not exceed 5 in the most complex modules, and in most modules it is 3 or less. Furthermore, the relatively simple data structures provided by HyperCard have been used to create simple objects and the equivalent of records and arrays, with no "tricky" or encoded data.

integrity

Integrity is determined by the detail criteria security and standards compatibility. Security has not been a major concern in developing a design for a proof of concept. The security of the hardware has not been considered at all. However, at least the basic security of the software must be addressed in an implementation. In the prototype, it was not deemed necessary to be concerned about protecting the software from any type of use, access, or disclosure. However, the software is protected from accidental modification or destruction in the sense that the casual user does not have the opportunity to perform any actions which can accidentally damage or destroy anything. All expected user actions either click a mouse on a button, click a mouse to choose an element from a list, or type in a number which is subsequently validated. Furthermore, the database

and knowledge base are never directly visible to the user. However, for this prototype, no attempt has been made to prevent the knowledgeable HyperCard user from examining the source code or the database and knowledge base (except where some of the source code has been turned into system resources through compilation, and then source code no longer exists in the prototype stack).

The design conforms to the IEEE standard guidelines used for requirements and design with one notable exception. The Ada program design language (PDL) used cannot be compiled using an Ada compiler. It would not require much modification to make the PDL compilable, but taking this step did not provide any additional advantage for the purposes of this research. Since the prototype was implemented in HyperTalk code, an Ada compiler was not used at all. The names used for identifiers in the prototype did not conform strictly with accepted Ada style. In Ada, an identifier name consisting of multiple words will typically have underscores between the words. However, although HyperTalk permits the use of underscores, it does not consider them as a part of the identifier name. This permits a potentially confusing inconsistent use of identifier names, where sometimes underscores may be used and sometimes they may be forgotten. Furthermore, when a HyperTalk script is printed, an underscore comes out as a blank. This is even more inconsistent and confusing. Hence, underscores were avoided in the HyperTalk code, and identifiers were written with the first letter of all words capitalized and all other letters in lower case. The PDL was written using the English words with regular spaces in

between, and they can then be easily translated into the appropriate form for either HyperTalk or Ada.

interoperability

Interoperability is determined by the detail criteria communication commonality, data commonality, modularity, rehostability, and retargetability. Communication commonality was not an issue in the ASSIST design because no communication is specifically defined. This could become an issue when the Data Acquisition Subsystem is more fully designed.

Data commonality deals with standard data structures and types which are used throughout the program. Again, the design is not very specific about how data structures and types are implemented, but there is nothing in the design which precludes the use of standard structures and types. The prototype uses structures and types which are predefined in HyperCard, and these are not standard to any other platform.

Modularity has already been discussed above under expandability.

Rehostability applies to an implementation rather than a design. It can be said that the design in no way precludes rehostability, but it also does not require it. Since the prototype is implemented in HyperCard, it is rehostable on any Macintosh which runs HyperCard, but not beyond that.

Since ASSIST is targeted to run on the same system which hosts its development, retargetability is the same as rehostability in this case.

maintainability

Maintainability is determined by the detail criteria augmentability, modularity, self documentation, simplicity, consistency, communicativeness, test availability, and structuredness. Augmentability, modularity, self documentation, and simplicity have already been discussed above under expandability, and consistency has been discussed under correctness. Therefore, only communicativeness, test availability, and structuredness are discussed here.

Communicativeness refers to the feedback provided by the program which it is operating. ASSIST requires that a reasonable degree of feedback be provided to keep the user informed of what is happening as the program executes. In the prototype, feedback is provided by both the spinning cursor and messages provided to user. A graphical browser also provides feedback on where the user is, conceptually, within the structure of the program activities.

In the realm of test availability, the ASSIST design includes a test plan. In an implementation, specific tests should be developed from this test plan for use in system testing. How available these tests would be with the acquisition of the software, however, would depend on the vendor.

The basic set of control structures (each having one entry point and one exit) have been used exclusively in both the ASSIST design and the prototype implementation.

reliability

Reliability is determined by the detail criteria accuracy, completeness, consistency, fault tolerance, modularity, and simplicity. Completeness and consistency have been discussed above under correctness. Modularity and simplicity have been discussed under expandability. Therefore, only accuracy and fault tolerance are discussed here.

Accuracy is a quantitative measure of the relative error in numerical calculations. In ASSIST the relative error will be high because of the imprecise nature of the numerical values used in the calculations. This is not a drawback in the ASSIST design, but rather a fact of life because of the subjective nature of software evaluation. ASSIST is no better or no worse than any other DSS in this area. However, unlike many applications of decision analysis, ASSIST is careful not to imply that a higher level of accuracy is possible.

Fault tolerance deals with a system's ability to continue after a limited number of hardware or software faults. Hardware failures and external (to ASSIST) software failures are not a particular concern for ASSIST. As far as ASSIST software failures are concerned, the design aims for a minimum number of possible failures by using lists and minimizing typed input. If a failure should occur, it could destroy the integrity of the current run of the

program. However, ASSIST can be restarted with no problem as long as no external software problems exist. For the prototype, HyperCard has proven to be a very stable software platform which should continue to execute even if ASSIST should hit a "glitch". HyperCard will easily restart the prototype.

reusability

Reusability is determined by the detail criteria application independence, generality, hardware independence, modularity, operating system independence, and self documentation. Generality, modularity, and self documentation are discussed above under expandability. Therefore, only application independence, hardware independence, and operating system independence are discussed here.

The ASSIST design is not dependent on the support required for any particular application, any particular hardware, or any particular operating system. However, the prototype is dependent on the HyperCard application, and it is currently dependent upon both the Macintosh hardware and operating system.

survivability

Survivability is determined by the detail criteria autonomy, distributedness, fault tolerance, modularity, and reconfigurability. Modularity is discussed above under expandability and fault tolerance is discussed under reliability. In general, survivability was

not considered to be important for a system such as ASSIST, which does not provide any type of critical functions.

The ASSIST design does not preclude autonomy in that it does not require external interfaces nor the use of external functions. However, if external database software were used in an ASSIST implementation, this would limit the autonomy of the system (unless standard interfaces were defined and used). The prototype is not very autonomous since it relies heavily on HyperCard interfaces and functions.

The ASSIST requirements do not specify distribution, and the design does not address the issue because it was not deemed important for the proof of concept. However, nothing would prevent the distribution of the elements of ASSIST. For example, the database could be distributed if a distributed database system were used in an implementation. The prototype is not distributed.

The ASSIST design does not provide for reconfigurability. If a processor or storage unit were to fail, the system would not be able to continue to operate.

transportability

Transportability is determined by the detail criteria hardware independence, modularity, operating system independence, rehostability, retargetability, self documentation, and support software independence. Modularity and self documentation are discussed above under expandability, rehostability and retargetability are discussed under interoperability, and hardware

independence and operating system independence are discussed under reusability. Therefore, only support software independence is discussed here.

The design is not dependent on any particular support software. However, an implementation will probably be dependent upon the database system it uses. The prototype is dependent upon HyperCard.

usability

Usability is determined by the detail criteria capacity, ease of installation, ease of use, maturity, on-line help, power, tailorability, and user documentation. Many of the usability criteria apply more to an implementation rather than a design. The ease of installation, power, tailorability, and user documentation are almost strictly implementation issues. The ASSIST requirements and design neither help nor hinder an implementation with respect to these criteria.

The ASSIST requirements do not limit the capacity of the database nor the knowledge base, and they specify that an implementation will not place an arbitrary limit on them. The available disk space on the system will provide the only limit.

The ASSIST requirements specify a number of system features which are aimed at ease of use. These range from providing suggested feature and criteria choices to a requirement for informative error messages. They include a requirement for the six support functions which are always available for use. These functions provide backtracking, a graphical browser, a review of the

current feature and criteria choices, printing, saving to and retrieving from disk, and on-line help. The issues involved in ease of use will be discussed more in conjunction with maturity in the next section on executing scenarios on the prototype. The maturity of ASSIST is, of course, minimal.

On-line help is required in ASSIST, and it is also required to be easily accessible. However, the usefulness of the help is still an implementation issue.

vendor support

Vendor support is determined by the detail criteria corporate health, pricing policies, reputation, and support policies. These criteria do not apply to the ASSIST design. They must be assessed separately for each vendor which should choose to implement the design. They also do not apply to the prototype since it is not for purchase from a vendor.

verifiability

Verifiability is determined by the detail criteria communicativeness, modularity, self documentation, simplicity, standards compatibility, structuredness, and test availability. All of these details have been discussed previously. Modularity, self documentation, and simplicity are discussed above under expandability, and standards compatibility is discussed under integrity. Communicativeness, structuredness, and test availability are discussed under maintainability.

Design Analysis

Now that the features and criteria of the ASSIST design have been evaluated, these assessments should be put in perspective. It is highly unlikely that any system could be designed to maximize all features and criteria. ASSIST is no exception, so this analysis examines how well it fares with respect to the characteristics deemed to be most important.

In the discussions of the system requirements and the system design, the objectives of ASSIST have been clearly defined. These objectives relate to certain specific features and criteria which have been defined in the ASSIST framework, so these features and criteria will now be enumerated.

The features which were treated as most important were the transformation functions, timing requirements, and user profile. ASSIST's transformation functions are those which deal with transforming input data into the internal form required by the database, and transforming the database data into a numerical form for the system calculations. These transformations were discussed in the previous chapter under the section Developing a Common Terminology. The part of the design dealing with the first transformation has not been completed as a part of this research effort. The transformation of data into numerical form has been addressed by the design, and all transformation mechanisms have been carefully isolated in the knowledge base to keep them independent of the remainder of the ASSIST system.

The timing requirements of concern are those which relate to system usability, and they deal with the system responses to various user actions. As has been discussed under *timing requirements* in the earlier discussion of features, the ASSIST requirements specify response times for all the various types of user actions. They also require that the user get feedback during periods of delay. Timing requirements is an area which must be continually considered as the prototype is further developed and as full implementations are completed. User feedback will help to determine if the current requirements are adequate, and if not, it will help to provide appropriate modifications to the requirements.

The target user profile for ASSIST is that the user needs little or no training and a skill level of novice. Once again, it remains to be seen if this has been accomplished, and user feedback will be very valuable in determining this. However, many aspects of the system design have addressed this issue in an attempt to provide a system which will require a minimal amount of user background and training. These have been discussed in detail in Chapter 4 on System Design. They are illustrated by such features as the use of very few keystrokes and the provision for a minimal path through the system which can be accomplished relatively quickly and easily and without knowledge of the technical details of software assessment. The scenarios in the next section help to illustrate how these features have been incorporated into the prototype.

The criteria which were the most important considerations in the ASSIST design were correctness, expandability, maintainability,

reliability, usability, and verifiability. The above discussions in each of these areas make it clear that each of the detail criteria which contribute to these categories has been carefully considered in the ASSIST design, and many of the detail areas overlap among the various important criteria. The details which fare the worst in these areas are accuracy and maturity. However, the poor accuracy of ASSIST is because of the imprecise nature of the numbers used in system calculations, and the lack of maturity is obvious in a system which has yet to be fully implemented. The poor quality of these criteria cannot be attributed to the ASSIST design.

Usability is, at the same time, one of the most important criteria against which to assess ASSIST and one of the most difficult to assess without having a full implementation of the design to work with. The next section attempts to help in this regard by providing a detailed discussion of two scenarios which have been executed on the ASSIST prototype. These scenarios are examples of how various parts of the system design may be implemented, although they do not necessarily represent every possible implementation of ASSIST.

A Case Study of the ASSIST Prototype

In order to execute the ASSIST prototype it was necessary to prepare the database with evaluation data. The data which went into the database were all for the evaluation of compilation systems. This was not a difficult exercise in terms of complexity, but it provided some new insights into the data acquisition process as well as the evaluation of compilation systems, and these are worthy of

discussion. The execution of the two scenarios will also be discussed at length, and the implications in terms of ASSIST's usability will be analyzed.

The Database for Selecting a Compilation System

It seemed reasonable to attempt to use data from actual evaluations of compilation systems in the prototype database. However, this proved to be a difficult task for two reasons.

First, many organizations were reluctant to release evaluation data of any type. It is often considered proprietary information. Even the government organization sponsoring this research was reluctant to release evaluation data because it is judgmental, and there was a fear that the correct names of products and vendors would be released to the public.

Second, after obtaining some of the normally well-guarded data, it became clear that it is not adequate for doing the job anyway. When trying to determine what evaluation data was required for ASSIST to make a "reasonable" comparison of software products, many types of data seemed to be necessary. However, the data provided by test suites which are available and used by many organizations actually satisfy only a very small part of the data requirements [44], [150].

Of course, each of the features and criteria provided by ASSIST is important to some people and organizations for some types of software selections. However, some of the features and criteria can be considered to be important most, if not all, of the time. Others are

important for particular types of software or particular application areas only.

To illustrate the features and criteria of most importance, lists were devised of the data which is almost always important to the acquisition process. These lists could be incorporated into forms which could be used to help with data acquisition for any ASSIST implementation.

Figure 5.1 illustrates a list of generic data which should be collected on features. This is data which should be provided in any software evaluation, regardless of the type of software or the application for which it is intended. For the most part, available test suites for evaluating compilation systems do not provide for the collection of feature information. However, much feature data can be obtained from running a test suite, either from using the system documentation or from observation. The problem is that the typical test suite gives little indication to the user that this information should be a part of any evaluation report produced after running the test suite. Some features, such as those dealing with product identification, are usually included in reports, but many assessors do not consider other important features covered in this list.

Figure 5.2 is a list of features which should always be reported when evaluating a compilation system. With the exception of compiling lines of code, most of these features also are not reported in the available test suites. However, some of the features may be discovered, or lower limits can be established because of the test suite execution or from information in the system documentation.

Product Feature Evaluation Data (Generic)	
Product identification:	
Product name and version	
Vendor name	
Configuration requirements:	
Host hardware	
Target hardware	
Operating system	
Minimum host primary memory	
Minimum host disk space	
Contractual matters:	
Number of users	
Number of CPUs	
Is support available	
Basic software price	
Installation costs	
Other costs	
Other required information:	
Applied standards	
Associated tool requirements	
Security level	
User skill level required	
User training required	
Functions supported	

Figure 5.1 - Important generic data to be collected on features

Figure 5.3 lists the criteria which are important to almost every software assessment. These are all relative measures, so in many ways they are the most difficult to assess. However, the typical test suite which evaluates the performance of an Ada compilation system actually provides more relative assessment measures than anything else. Even so, the data is usually restricted to the determination of processing effectiveness and storage effectiveness.

Product Feature Evaluation Data Compilation Systems	
Source code sizing:	
	Max lines per compilation unit
	Max units in a compile
	Max entries in a task
	Max alternatives in a case
	Max alternatives in a select
	Max instantiations of a generic
	Max elements in an aggregate
	Max discriminants in a record
Numerics:	
	Bits in an integer
	Bits in a float
	Fixed point delta
	Support for long rep forms
	Support for short rep forms
Timing:	
	Compiling lines of code (lines/min)
	Task rendezvous overhead
	Subprogram overhead
	Exceptions overhead
	Clock resolution
	Max blocking time

Figure 5.2 - Important data to be collected on features specific to compilation systems

Of course, the data which comes from the test suites discussed includes much detailed data in addition to the data listed in the figures presented here. Bear in mind that this detail can be handled by ASSIST, but the higher level of abstraction shown in the figures is also an important perspective. The detail data can be of interest to the technically-oriented decision maker at a lower level of abstraction. It can be very valuable in the selection of an Ada compilation system for certain applications, but it requires a knowledge of statistics to be able to use it effectively. The ACEC test

suite provides analysis tools which report the statistics of interest, but someone still needs to know how to interpret them [44], [110], [141].

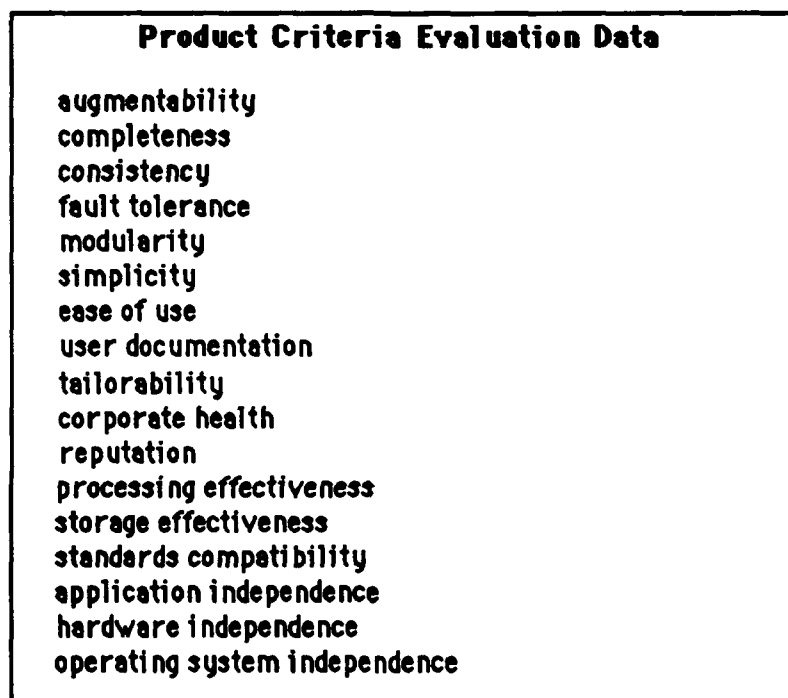


Figure 5.3 - Important data to be collected on criteria

The problem in trying to populate the prototype database seemed to be twofold. First, assessors do not usually report everything they could. Presumably this is either because they had no interest in other things or because they did not think about other types of assessment while reporting on the results of executing a test suite. Secondly, the assessment tools are not as comprehensive as they could be. The test suites which have been examined do a reasonably good job in providing specific types of performance tests

for Ada compilation systems. However, they do not attempt to provide other assessment information, nor do they even provide suggested checklists, such as Figures 5.1 through 5.3, which could be used in conjunction with the execution of the test suite. Other helpful tools, such as Weiderman's *Ada Adoption Handbook: Compiler Evaluation and Selection* [161] provide valuable information to the potential assessor about what information to collect and how to avoid common pitfalls. Weiderman's handbook, in particular, is comprehensive in its coverage. However, it is targeted to the assessor who will do everything in-house, so it does not discuss such matters as how to prepare an assessment report and what to include in it. Too often, when assessment reports are prepared, they get so bogged down in the detail that they lose the higher level perspective.

A good source for checklists which may be used to collect feature details for the ASSIST database is the *E&V Guidebook* [49]. A number of other sources, such as Weiderman's handbook mentioned above, provide guidelines for collecting assessment data [61], [80], [97], [104], [161], [171]. However, the *E&V Guidebook* is the only one providing actual checklists which may be used as is. It also provides pointers to most other sources of evaluation technology, and its companion document, the *E&V Reference Manual* [50], includes a comprehensive discussion of attributes, which are called criteria in ASSIST. As software evaluators begin to use these relatively recent documents, a broader range of assessment data should become available for ASSIST.

Because it was so difficult to obtain actual evaluation data on so many of the important features and criteria of a compilation system, the prototype database contains much composite data. This data represents reasonable expectations from compilation systems currently on the market. It also includes some embellishments which were added for the specific purpose of exercising the capabilities of the prototype.

Once the database was populated with assessment data, scenarios could be run. Two scenarios were chosen to illustrate the capabilities of the prototype. The first is a likely scenario for a management-oriented decision maker, and the second is more likely for a technically-oriented decision maker.

The Management-Oriented Scenario

As this scenario begins, ASSIST is already started, and the first window is visible. The decision maker has the User Manual (see Appendix L) available for reference. As the scenario progresses, each window is referred to by the name visible in the window. Each action taken by the decision maker is described, along with what occurs on the screen in response to the action. Appendix M illustrates the windows seen by the user (sometimes in various states) during the execution of this scenario. This discussion refers to these illustrations to help clarify what the user sees.

Action 1. The user clicks on the "Click Here" button in the title window (see top of page 1 in Appendix M). A small box appears which covers the "Click Here" button, and it says "The system is

initializing...". The cursor, which normally looks like a small hand, turns into a spinning ball to indicate that the system is working. After a few seconds, the Explanations window appears (see bottom of page 1 in Appendix M) and the cursor stops spinning and goes back to the small hand.

Action 2. The user decides to click on the Compass icon to see what it does. This immediately brings the Compass Icon window into view (see top of page 2 in Appendix M).

Action 3. A click on the "Graphic" button brings the Where Am I window into view (see bottom of page 2 in Appendix M). The user studies this for a few seconds to be sure she understands the process used by the program.

Action 4. The user clicks on the Return arrow and gets back to the Explanations window.

Action 5. The user decides to move on, and she clicks on the "OK" button. This brings the Context Choices window into view (see top of page 3 in Appendix M).

Action 6. After looking through the User Manual, the user clicks on "Tool Set" to get to the selection of a compilation system. This brings up the Tool Set window (see bottom of page 3 in Appendix M).

Action 7. A click on "Compilation system" brings the Application Area window into view (see top of page 4 in Appendix M).

Action 8. This decision maker is interested in an Ada compilation system to use in developing a management information

system, so she clicks on "Information intensive". After a short pause, this brings up the Choose Features window.

Action 9. The user sees a list of five suggested general feature choices. She chooses "contractual matters", "cost", and "user profile" by clicking in the box to the left of each choice. An "x" shows in the box for each chosen feature (see bottom of page 4 in Appendix M).

Action 10. Before going on, the user decides to see if she may find any other feature categories of interest. She clicks on "Add Feature" and two boxes appear in the window (see top of page 5 in Appendix M). The box at the left contains a list of all feature categories from which she may choose, and the box at the right contains an explanation of how to make a choice.

Action 11. The user decides to click on the "Cancel" button and make no additional choices. This makes the small boxes disappear, and the Choose Features window once again shows the three choices which have been made.

Action 12. A click on "OK" turns the cursor into a spinning ball, and a small box appears for a few seconds which explains that feature details will now be processed (see bottom of page 5 in Appendix M). Then a Choose Feature Details window appears. This is a window for specifying the details for "contractual matters".

Action 13. The user chooses "no restrictions on users" and "support available" from the given list by clicking in the box next to each (see top of page 6 in Appendix M).

Action 14. The user clicks on "OK". This turns the cursor into a spinning ball, and the Choose Features window appears again with

the box still showing which explains the detail feature processing. A magnifying glass is now covering the check box beside "contractual matters". Another small box appears toward the top of the window with a message which says, "Specify cost as $\leq$ (U.S. Dollars)", with a highlighted line below it containing a suggested figure of 10,000.

Action 15. The user clicks on "OK", accepting the 10,000 figure. After another pause with the spinning cursor, another Choose Feature Details window appears. This one is for specifying details concerning "user profile".

Action 16. The user chooses both "skill level" and "training" by clicking in the box next to each.

Action 17. The user clicks on "OK". Another specification box appears which says, "For skill level choose from the following", with three buttons indicating three possible choices for the highest expected skill level, "expert", "intermediate", or "novice" (see bottom of page 6 in Appendix M).

Action 18. The user clicks on "intermediate", the box immediately disappears and "intermediate" appears beneath "skill level" in the choice list. Another specification box appears for choosing the maximum amount of expected training, with choices of "extensive", "moderate", and "little".

Action 19. The user clicks on "moderate", the box disappears and "moderate" appears beneath "training" in the choice list (see top of page 7 in Appendix M). The cursor starts spinning and the Choose Features window appears again. The feature detail explanation box disappears, and a magnifying glass is covering the check box next to

"user profile". A review message has also replaced the original explanation message in the window, and a "Done" button has replaced the "OK" button. The specification " ≤ 10000 U.S. Dollars" is immediately below "cost" in the choice list (see bottom of page 7 in Appendix M).

Action 20. The user clicks on "Done", and after a few seconds the Weight Features window appears. It contains a list of the three chosen general feature categories with a default weight of 5 beside each.

Action 21. The user clicks on the 5 next to cost, and a small specification box appears again. This time it says, "Type in value for weight between 0 & 10", and the old weight 5 is highlighted on the line below.

Action 22. The user types "10" and this replaces the highlighted value.

Action 23. The user clicks on "OK" in the box, the box goes away, and the new value of 10 replaces the old value of 5 next to "cost" (see top of page 8 in Appendix M).

Action 24. The user clicks on "OK", and a box appears which explains that weights will be considered for each of the detail features (see bottom of page 8 in Appendix M). The cursor spins, and after a few seconds, a Weight Feature Details window appears. It lists the detail choices made for contractual matters, along with a default weight of 5 for each detail choice.

Action 25. A click on "OK" accepts the default weights. The cursor spins, and after a few second another Weight Feature Details

window appears. This is for user profile, and it contains the detail features chosen for user profile, along with a default weight of 5 for each (see top of page 9 in Appendix M).

Action 26. The user clicks on "OK" again to accept the default weights. The Weight Features window appears, and now a magnifying glass appears to the left of each feature category for which details have been specified (see bottom of page 9 in Appendix M).

Action 27. The user clicks on "Done". After a delay of a few seconds with the cursor spinning, the Choose Criteria window appears. It contains suggested criteria.

Action 28. The user clicks in the check boxes next to "correctness", "usability", and "vendor support" (see top of page 10 in Appendix M).

Action 29. The user decides to see what other criteria are available, and clicks on "Add Criterion". This brings into view a box containing other criteria choices, and another box explaining how to make a choice.

Action 30. The user chooses "reliability" by clicking on it. The word is momentarily highlighted, then the small boxes disappear and a new check box is added to the choice list, along with the word "reliability" next to it (see bottom of page 10 in Appendix M).

Action 31. The user clicks on "OK", and a box appears with the question "Specify details for these criteria?"

Action 32. The user is not quite sure of the implications of this, and she attempts to get help by clicking on the question mark.

However, this does not work, and she gets a "beep" from the computer.

Action 33. The user clicks on "No", and the window for Weight Criteria appears. It contains a list of the chosen criteria with a default weight beside each.

Action 34. The user still wants to know the implications of choosing no details for the criteria, so she clicks on the Help question mark again. This time it works and the Help window for Weight Criteria appears (see top of page 11 in Appendix M).

Action 35. Since she really wants to know something about choosing criteria, the user clicks on "Choose Criteria", and the Help window for Choose Criteria appears (see bottom of page 11 in Appendix M).

Action 36. After reading the information in the window, the user still does not have her question answered, so she clicks on "No Details". This brings the Help window for No Details into view (see top of page 12 in Appendix M), and it answers her question.

Action 37. Now that she is satisfied that the criteria details will be handled appropriately, the user clicks on "Quit Help". The window for Weight Criteria appears again (see bottom of page 12 in Appendix M).

Action 38. The user clicks on "OK" to accept the default weights for the criteria. The cursor spins and the Display Information window comes into view with a box containing the message "Please wait ..." (see top of page 13 in Appendix M). A message box at the bottom of the screen also gives periodic progress messages as the

system takes several minutes to process all the data in its database. Eventually, the wait message disappears in the Display Information window, and a list of software product recommendations appears (see bottom of page 13 in Appendix M).

Action 39. The user scrolls through the information in the window by clicking on the arrow at the bottom of the scroll bar. She sees that six compilation system products were considered, and four were deemed acceptable while two were unacceptable, based on the features and criteria she had chosen. (See top of page 14 in Appendix M for the unacceptable list.)

Action 40. The user is satisfied with the results and decides to print the window containing the top three products listed in the recommendations. With the window scrolled all the way to the top again, she clicks on the Printer button. The Print Selection window comes into view (see bottom of page 14 in Appendix M).

Action 41. The user clicks on "Print Window", and the Display Information window appears again. A message box appears to inform her that the window is being printed. After a few seconds, the message box disappears again and the window is printed.

Action 42. The user clicks on "Quit ASSIST", and a small box appears with the question "Save chosen features and criteria?".

Action 43. A click on "No" starts the cursor to spin as a message box appears saying, "Cleaning up". After a few seconds, ASSIST is done executing.

This concludes the management-oriented scenario. An analysis of this scenario will be made along with an analysis of the

technically-oriented scenario. These analyses will follow the description of the technically-oriented scenario.

The Technically-Oriented Scenario

Once again, as this scenario begins, the first window (the title window) of ASSIST is visible, and the decision maker has the User Manual available for reference. Many of the windows seen by this user are the same as those illustrated in Appendix M for the management-oriented scenario already described. Appendix N illustrates windows which are different for this scenario.

Action 1. The user clicks on "Click Here", gets the system initialization message, and after a few seconds the Explanations window appears.

Action 2. A click on "OK" makes the Context Choices window become visible.

Action 3. This decision maker clicks on "Life Cycle Activity" because he wants to consider a compilation system for developing a real time simulation. He knows that a compilation system is a part of the life cycle area of implementation. The Life Cycle Activity window appears (see top of page 1 in Appendix N).

Action 4. The user clicks on "Implementation", and this brings a message box which says, "For now, this is a compilation system only".

Action 5. A click on "OK" makes the Application Area window appear.

Action 6. The user clicks on "Soft real time". After the cursor spins for a few seconds, the Choose Features window appears with a list of suggested features.

Action 7. The user clicks in the check boxes next to "source code sizing" and "timing requirements". This results in an "x" in each of these check boxes.

Action 8. The user clicks on "Add Feature", and a list of additional features appears along with an explanation of how to choose a feature from the list.

Action 9. The user clicks on "configuration requirements", and one of the words highlights briefly. Then the boxes containing the list of features and the explanation disappear, and a new check box appears with "configuration requirements" next to it (see bottom of page 1 in Appendix N).

Action 10. A click on "OK" starts the cursor spinning, and a box appears with the explanation that feature details will now be specified. After a short delay, the Choose Feature Details window appears with a list of details which may be chosen for source code sizing.

Action 11. The user clicks in the boxes next to "lines in unit", "units in compile", "entries in task", and "instantiations of generic" (see top of page 2 in Appendix N).

Action 12. The user clicks on "OK". A box appears with the message "Specify lines in unit as >=", and the line below this contains a highlighted default value of 5000.

Action 13. The user clicks on "OK", accepting the default value. The message box goes away, and the specification " ≥ 5000 " appears immediately below "lines in unit". Then another message box appears asking to specify units in compile with a default of ≥ 200 .

Action 14. The user types "500", and this replaces the default value in the message box.

Action 15. The user clicks on "OK", and the specification " ≥ 500 " appears immediately below "units in compile". Then another message box appears for the specification of entries in task.

Action 16. The user clicks on "OK" to accept the default value of ≥ 20 , and this specification appears with "entries in task". Another message box appears for the specification of instantiations of generic.

Action 17. The user clicks on "OK" to accept the default value of ≥ 10 , and this specification appears with "instantiations of generic". The message box is now gone. The cursor spins and the Choose Features window again appears with the message about specifying detail features. Then the Choose Feature Details window appears again, and this time it contains a list of details for timing requirements.

Action 18. The user clicks in the check boxes next to "task rendezvous" and "max blocking time".

Action 19. The user clicks on "OK", and the message box appears for specification of a task rendezvous time.

Action 20. A click on "OK" accepts the default specification of ≤ 0.00001 sec. This value appears below "task rendezvous", and a new message box appears for specification of max blocking time.

Action 21. The user clicks on "OK" to accept the default specification of ≤ 0.0002 sec. The message box disappears and the specified value appears below "max blocking time". The cursor spins and the Choose Features window appears again with the detail specification message. Then the Choose Feature Details window appears again, and this time it contains a list of details for configuration requirements.

Action 22. The user clicks in the check boxes next to "host memory needed", "host disk capacity needed", and "operating system".

Action 23. The user clicks on "OK", and the message window appears for specifying host memory needed.

Action 24. The user clicks on "OK" to accept the default value of ≤ 4 MB. This value appears below "host memory needed", and a new message box appears for specification of host disk capacity needed.

Action 25. A click on "OK" accepts the default value of ≤ 50 MB. This value appears below "host disk capacity needed". Two new boxes now appear. One contains the list of all operating systems in the database, and the other contains an explanation of how to choose one.

Action 26. The user clicks on "VMS", and this value appears immediately below "operating system". The cursor spins and the

Choose Features window reappears. Now the detail specification message goes away and magnifying glasses are covering the check boxes beside the chosen feature categories (see bottom of page 2 in Appendix N).

Action 27. The user decides to review the choices he made for source code sizing, so he clicks on the magnifying glass next to "source code sizing". The Choose Feature Details window for that feature again appears (see top of page 3 in Appendix N).

Action 28. Since he decides not to change anything, the user clicks on the Return arrow to get back to the Choose Features window, and that window reappears.

Action 29. The user now clicks on "Done", and the Weight Features window appears. It contains a list of the chosen feature categories with suggested weights listed next to each (see bottom of page 3 in Appendix N).

Action 30. The user clicks on "OK" to accept the default weights. The cursor spins and a message appears indicating that the detail weights will now be processed. After a few seconds, the Weight Feature Details window appears with the list of all details chosen for source code sizing, along with default weights of 5 for each detail.

Action 31. The user clicks on "units in compile", and the entire line containing it and its default weight is highlighted. Then a message box appears asking for a new weight to be specified.

Action 32. The user types in "10", and this replaces the highlighted 5 which had been in the message box.

Action 33. The user clicks on "OK", the message box goes away, and the new weight of 10 replaces the 5 beside "units in compile" (see top of page 4 in Appendix N).

Action 34. The user clicks on "OK", and the Weight Feature Details window appears with the choices which were made for timing requirements, along with default weights of 5 for each.

Action 35. The user clicks on "task rendezvous" to change its weight, the line highlights, and the message box appears for the weight specification.

Action 36. The user types "10", and this shows in the message box.

Action 37. The user clicks on "OK", the message box goes away, and the weight of task rendezvous is changed to 10.

Action 38. The user clicks on "OK", and the Weight Feature Details window appears with the choices which were made for configuration requirements, along with default weights of 5 for each.

Action 39. The user clicks on "OK" to accept the default weights. The Weight Features window appears again, and magnifying glasses are next to each feature category listed (see bottom of page 4 in Appendix N).

Action 40. The user clicks on "Done", the cursor spins, and after a short delay the Choose Criteria window appears. It contains a list of suggested criteria.

Action 41. The user clicks in the check boxes next to "correctness", "efficiency", "integrity", and "reliability" (see top of page 5 in Appendix N).

Action 42. The user clicks on "OK", and a message box appears asking if details should be specified for the criteria.

Action 43. The user clicks on "Yes", and the message box goes away. The cursor spins and another message appears indicating that details will be specified for each criteria. After a few seconds, a Choose Criteria Details window appears. It contains a list of the detail criteria for correctness.

Action 44. The user clicks in the check boxes beside "completeness" and "traceability" (see bottom of page 5 in Appendix N).

Action 45. The user clicks on "OK". The Choose Features window appears again briefly with the detail specification message still showing. Then the Choose Criteria Details window appears with a list of details which could be chosen for efficiency.

Action 46. The user clicks in the check boxes next to "processing effectiveness" and "storage effectiveness".

Action 47. The user clicks on "OK". The Choose Features window again appears with the detail message, and then the Choose Criteria Details window appears with a list of details for integrity.

Action 48. The user is surprised by the fact that only "security" and "standards compatibility" are listed for integrity. He figures that he may not understand the definition used here for integrity, so he decides to look at the definition. He clicks on "Definitions", and the Definitions window appears (see top of page 6 in Appendix N).

Action 49. The user scrolls through the criteria list until he finds "integrity".

Action 50. The user clicks on "integrity" and the definition appears in the middle of the window (see bottom of page 6 in Appendix N).

Action 51. The user is satisfied that he misunderstood what integrity meant, so he clicks on "OK" and the definition box goes away.

Action 52. The user does not want to look at any more definitions, so he clicks on the Return arrow. The Choose Criteria Details window reappears.

Action 53. Since he no longer wants to choose integrity, the user clicks on "OK" without having any details chosen. A message box appears with a reminder that no details have been chosen and asking if that is OK.

Action 54. The user clicks on "OK", and the message box disappears. The Choose Criteria window again appears briefly with the detail message still showing, but integrity is no longer checked. After a short pause, the Choose Criteria Details window appears again. This time it lists details for reliability.

Action 55. The user clicks in the check boxes next to "accuracy", "consistency", and "fault tolerance".

Action 56. A click on "OK" and the Choose Criteria window reappears. This time the details message disappears, and now each chosen criteria has a magnifying glass covering its check box (see top of page 7 in Appendix N).

Action 57. The user clicks on "Done", and the Weight Criteria window appears. It contains a list of the chosen criteria categories along with suggested weights.

Action 58. The user clicks on "reliability" to change its weight. The message box appears for specifying weight.

Action 59. The user types in "7" as the new weight.

Action 60. The user clicks on "OK", and the message box disappears. The weight is changed (see bottom of page 7 in Appendix N).

Action 61. The user clicks on "OK", and a message appears briefly indicating that detail weights will be processed. The Weight Criteria Details window then appears with a list of the choices for correctness, along with default weights of 5.

Action 62. The user clicks on "completeness" to change its weight, and the message box appears to ask for the new weight.

Action 63. The user types "9" as the new weight.

Action 64. The user clicks on "OK", and the message box disappears. The weight is changed.

Action 65. The user clicks on "OK", and the Weight Criteria Details window appears with a list of the choices for efficiency, along with default weights of 5.

Action 66. The user clicks on "processing effectiveness" to change its weight, and the message box appears to ask for the new weight.

Action 67. The user types "8" as the new weight.

Action 68. The user clicks on "OK", and the message box disappears. The weight is changed.

Action 69. The user clicks on "OK", and the Weight Criteria Details window appears with a list of the choices for reliability, along with default weights of 5.

Action 70. The user clicks on "fault tolerance" to change its weight, and the message box appears to ask for the new weight.

Action 71. The user types "10" as the new weight.

Action 72. The user clicks on "OK", and the message box disappears. The weight is changed.

Action 73. The user clicks on "OK", and the Weight Criteria window reappears. A magnifying glass is now beside each chosen criterion (see top of page 8 in Appendix N).

Action 74. The user clicks on "Done". The cursor spins and the Display Information window appears with a "Please wait..." message. A message box at the bottom of the screen also gives periodic progress messages as the system takes several minutes to process all the data in its database. Eventually, the wait message disappears in the Display Information window, and a list of software product recommendations appears.

Action 75. The user scrolls through the information in the window. He finds four compilation system products recommended as acceptable, one was unacceptable, and another did not meet all the feature requirements.

Action 76. The user decides to look at more details concerning the recommendations, so he clicks on the magnifying glass. A wait

message appears for a few seconds. Then a description of the system calculations appears in the Display Information window (see bottom of page 8 in Appendix N).

Action 77. The user scrolls through this information. After the information on the calculations, he finds a list of the current system parameters followed by a list of the numerical ratings calculated for each product considered (see top of page 9 in Appendix N).

Action 78. The user decides to print the list of recommended software along with their numerical ratings. Leaving this information in the window, he clicks on the Printer button. This brings up the Print Selection window.

Action 79. A click on "Print Window" and the Display Information screen appears along with a brief print message. The window is printed.

Action 80. The user clicks on the Eye button to find out what type of missing information is described. A wait message appears briefly, and then the Display Information window displays a description of why missing information is important (see bottom of page 9 in Appendix N).

Action 81. The user scrolls through the information in the window. The description is followed by a list of each software product considered, along with the features and criteria for which the product has no data in the database (see top of page 10 in Appendix N). He discovers that traceability and accuracy are missing for all the products, and the desired operating system is missing for

all but one of the products. Since all the products have been rated on the same information, he decides that the results are valid.

Action 82. The user decides to do a sensitivity analysis on the results. He clicks on "Change Choices", and the Choose Features window appears again. It still shows the choices which were made.

Action 83. The user clicks on the magnifying glass next to "configuration requirements", and the Choose Feature Details window appears with the configuration requirements choices.

Action 84. The user unchooses all three of the choices which had been made in this category by clicking on the check boxes next to each.

Action 85. The user clicks on "OK". A message appears which asks if it is OK that no choices have been made.

Action 86. The user clicks on "OK", and the Choose Features window reappears. Now it has no magnifying glass or check beside "configuration requirements" (see bottom of page 10 in Appendix N).

Action 87. The user clicks on "OK", and a message appears asking if the previously chosen features should be replaced.

Action 88. The user clicks on "Yes". The message disappears, the cursor spins, and after a few seconds a new message appears indicating that details will now be specified. The Choose Feature Details window appears. It contains the original list of all possible detail choices for source code sizing, and nothing is chosen. Everything must be respecified.

Action 89. The user clicks in the boxes next to "lines in unit", "units in compile", "entries in task", and "instantiations of generic".

Action 90. The user clicks on "OK". A box appears with the message "Specify lines in unit as $\geq$ ", and the line below this contains a highlighted default value of 5000.

Action 91. A click on "OK" accepts the default value. The message box goes away, and the specification " $\geq$ 5000" appears immediately below "lines in unit". Then another message box appears asking to specify units in compile with a default of $\geq$ 200.

Action 92. The user types "500", and this replaces the default value in the message box.

Action 93. The user clicks on "OK", and the specification " $\geq$ 500" appears immediately below "units in compile". Then another message box appears for the specification of entries in task.

Action 94. The user clicks on "OK" to accept the default value of $\geq$ 20, and this specification appears with "entries in task". Another message box appears for the specification of instantiations of generic.

Action 95. The user clicks on "OK" to accept the default value of $\geq$ 10, and this specification appears with "instantiations of generic". The message box is now gone. The cursor spins and the Choose Features window again appears with the message about specifying detail features. Then the Choose Feature Details window appears again, and this time it contains a list of details for timing requirements.

Action 96. The user clicks in the check boxes next to "task rendezvous" and "max blocking time".

Action 97. The user clicks on "OK", and the message box appears for specification of a task rendezvous time.

Action 98. The user clicks on "OK" to accept the default specification of ≤ 0.00001 sec. This value appears below "task rendezvous", and a new message box appears for specification of max blocking time.

Action 99. The user clicks on "OK" to accept the default specification of ≤ 0.0002 sec. The message box disappears and the specified value appears below "max blocking time". The cursor spins and the Choose Features window reappears. This time the "Done" button is visible instead of the "OK" button (see top of page 11 in Appendix N).

Action 100. The user clicks on "Done", and the Weight Features window appears. It contains a list of the chosen feature categories with default weights listed next to each. The magnifying glasses are gone.

Action 101. The user clicks on "OK" to accept the default weights. The cursor spins and a message appears indicating that the detail weights will now be processed. After a few seconds, the Weight Feature Details window appears with the list of all details chosen for source code sizing, along with default weights of 5 for each detail.

Action 102. The user clicks on "units in compile", and the entire line containing it and its default weight is highlighted. Then a message box appears asking for a new weight to be specified.

Action 103. The user types in "10", and this replaces the highlighted 5 which had been in the message box.

Action 104. The user clicks on "OK", the message box goes away, and the new weight of 10 replaces the 5 beside "units in compile".

Action 105. The user clicks on "OK", and the Weight Feature Details window appears with the choices which were made for timing requirements, along with default weights of 5 for each.

Action 106. The user clicks on "task rendezvous" to change its weight, the line highlights, and the message box appears for the weight specification.

Action 107. The user types "10", and this shows in the message box.

Action 108. The user clicks on "OK", the message box goes away, and the weight of task rendezvous is changed to 10.

Action 109. The user clicks on "OK". The Weight Features window appears again, and magnifying glasses are next to each feature category listed.

Action 110. The user clicks on "Done", the cursor spins, and the Choose Criteria window appears. The original criteria specifications are still shown.

Action 111. The user clicks on the Review button to see what the current specifications are. The Review window appears (see bottom of page 11 in Appendix N).

Action 112. The user scrolls through the information in this window. He finds information on the type of software, the

application area, the chosen features and weights assigned to each, the chosen criteria and weights assigned to each, and the system parameters and their current values.

Action 113. The user clicks on Return, and the Choose Criteria window appears again.

Action 114. The user clicks on the Compass button because he does not want to change any of the criteria specifications. The Compass Window appears with Choose Criteria highlighted (see top of page 12 in Appendix N).

Action 115. The user clicks on "Display Information", and the Display Information window appears.

Action 116. The user clicks on "New Display". The "Please wait..." message appears for a few minutes as the data is processed. Then the message goes away and a new display of recommendations appears.

Action 117. The user scrolls through the information and finds that the same four products have been recommended again. The only difference is that the order of the third and fourth products in the list has been switched.

Action 118. The user wants to see details again, so he clicks on the magnifying glass and a please wait message appears. After a short delay, the Display Information window appears with the detail information on calculations.

Action 119. The user scrolls through the information to see how much the ratings have changed. He finds that they are slightly different from the last time, but not a lot.

Action 120. The user decides to do one more sensitivity check. He clicks on "System Parameters", and the current values of the system parameters appear in the Display Information window.

Action 121. The user clicks on "Change Parameters", and a message box appears asking for the value of the overall weight for the features. The current value of 1 is given as a default.

Action 122. The user clicks on "OK" to accept the current value. The message box now asks for the value of the overall weight for the criteria. The current value of 1 is given as a default.

Action 123. The user types "2", and this replaces the default value.

Action 124. A click on "OK" accepts this new value. The message box now asks for the value of the minimum acceptable rating. The current value of 0.9 is given as a default.

Action 125. The user types "1.0", and this replaces the default value.

Action 126. The user clicks on "OK" to accept this new value. The message box goes away, and the new values for the system parameters appear in the window (see bottom of page 12 in Appendix N).

Action 127. The user clicks on "New Display". The information in the window is cleared, and the "Please wait..." message appears for a few minutes as the data is processed. Then the message goes away and a new display of recommendations appears.

Action 128. The user scrolls through the information and finds that only the top two of the previously recommended products have been recommended this time.

Action 129. The user wants to see details again, so he clicks on the magnifying glass and a wait message appears. After a few seconds the detail information on calculations appears in the window.

Action 130. The user scrolls through the information to see how much the ratings have changed. He finds that again they have not changed that much. The results consistently show one product with a significantly higher rating than any of the others. The second acceptable product is on the borderline. The third and fourth products are now slightly below the cut-off for acceptable.

Action 131. The user is now satisfied with the results, and he wants to save his specifications. He clicks on the Disk button. The Save or Retrieve Choices window appears (see top of page 13 in Appendix N).

Action 132. The user clicks on "Save". The cursor spins and a please wait message appears. After a few seconds, the Display Information window reappears.

Action 133. The user clicks on "Quit ASSIST". The message "Save chosen features and criteria?" appears.

Action 134. The user clicks on "No". The cursor spins for a few seconds and a message box appears which says "Cleaning up". Then ASSIST is done executing.

Scenario Analyses

The scenarios have been described to help evaluate the potential usability of an ASSIST implementation. It must be noted that other implementations could make the user interface look very different. Nevertheless, this discussion provides a reasonable assessment of the effect the design specifications will have on the usability of an ASSIST system. It also points out the places where the implementation could be improved.

Each action described in each scenario is a single click of the mouse (the only exception is where clicking in multiple check boxes is considered a single action), a scrolling action, or typing a single response. This makes the description as primitive and complete as possible.

In the scenarios, features, criteria, and their respective weights have been specified. Criteria have been specified both with and without details. All support functions have been used, as well as many other functions such as adding features and criteria and getting definitions.

It seems clear from an examination of the scenarios that a great deal of specification and examination was accomplished in a relatively small number of actions. A breakdown is provided in Figure 5.4 which makes this even more clear.

In the relatively straightforward management-oriented scenario, features and criteria were chosen only once. However, this still included the specification of 7 features (3 requiring the specification of values), 3 criteria, and weights for all 10 of the

characteristics. All of this was specified, and other actions were performed as well, with only 47 mouse clicks, 2 keystrokes, and 1 scrolling action.

Scenarios	Management-oriented	Technically-oriented
Number of actions	43	134
Number of mouse clicks	47	132
Feature choices	7	18
Feature value choices	3	15
Criteria choices	3	10
Weight choices	10	30
Support functions used	3	5
Number of keystrokes	2	23
Feature value specifications	0	2
Weight specifications	1	8
Number of scrolls	1	9

Figure 5.4 - Breakdown of scenario actions

The technically-oriented scenario was more complex than the first. It included an initial specification of 12 features (9 requiring values), 10 criteria, and weights for all 22 of the characteristics. Of these, 1 of the feature values and 6 of the weights required numerical values to be input from the keyboard. Then, 6 feature details were specified over again (all requiring values), and all 8 feature weights were chosen again. Even so, all of this and more was accomplished with only 132 clicks of the mouse, 23 keystrokes, and 9 scrolling actions.

In each case, the only things which had to be typed were numbers. All other choices could be made from lists just by clicking the mouse on the choice in the list. This indicates a relatively high level of usability.

However, everything was not so good that improvements are not possible and desirable. In addition to some parts of the ASSIST design that are just not implemented, some of the features in the prototype are clearly suffering from immaturity. For example, in Action 32 of the first scenario, the user was not able to get to help while a message box requiring a response was visible. She had to respond to the question first. Yet she wanted help so she would better understand how to respond to the question.

Another example of the prototype immaturity shows up starting at Action 88 in the second scenario. At this point some features had been deleted, but the remaining features were still fully specified. However, the action of changing the feature specification to reflect the deleted features resulted in all detail features having to be respecified from scratch. All feature weight specifications had to be redone as well. This accounted for 19 extra actions which had to be unnecessarily repeated. Not only is this repetitious activity, but it detracts significantly from the user's ability to perform sensitivity studies on the database.

A third example of immaturity in the prototype occurs near the end of the second scenario. In Action 133, the user is asked if the chosen features and criteria should be saved before quitting.

However, they were just saved in the previous action. Nothing could have changed, so another save is redundant.

All of these examples of immaturity point to the need for more sophistication in the prototype as well as in a full implementation. However, they do not give any indication that the high level ASSIST design needs to be changed. It is the detailed design which needs more work in these areas to make the system usability even better.

Final Analysis

To conclude this research effort, a number of users were invited to use the final prototype, and they were requested to fill in the questionnaire illustrated in Appendix O. Again, an effort was made to include both management-oriented and technically-oriented decision makers in this final evaluation. The results were once again generally positive, and criticisms centered on features which need to be added to the knowledge base. Both technical and non-technical users found the system very easy to use. The most positive response was from a technically-oriented user who just wants the database to be filled with "real" data so the prototype can be used as a productive system in the regular work place.

To go back to the original purpose of this research, ASSIST was conceived as a DSS which would provide a decision maker with useful recommendations concerning software selection decisions. At this point, it is time to look at whether or not this goal has been achieved.

The ASSIST design allows for an extendable knowledge structure to ensure that all aspects of technical software evaluation can be included in the database and used in recommendations. It also provides an initial knowledge framework as a point of departure for including the important aspects of an evaluation. The design has accounted for recommendations to the user in the form of acceptable products, the calculations used and the results of the calculations, and missing information which may have adversely affected the calculated results. Furthermore, the usability of the system has been addressed in both the design and the prototype. Hence, the ASSIST design provides for a system which can appropriately organize and summarize all of the available evaluation information on APSEs and present it to a decision maker in an effective and timely manner. The thesis of this research has been accomplished.

The next chapter will conclude by summarizing the research results and presenting a look at the potential impact of this research on the future of software evaluation technology.

6. Conclusions and Recommendations

The research described in this paper has been a start, but only a start, toward developing a comprehensive APSE assessment capability. The current state of assessment technology has been examined, and it has provided the basis for the evaluation framework which has been designed. This framework is an important and necessary step in determining the type of assessment data which must be provided in order to be able to make a comprehensive software evaluation. Designing a prototype computer system for APSE selection has demonstrated the viability of using this framework to improve the software selection process. However, much remains to be done before this framework can contribute to the state of the practice in software selection.

Accomplishments

The ASSIST system was designed in the form of a DSS. It provides for entering assessment data into its database and then using this data to provide comprehensive, yet concise and understandable, information to the decision maker in the form of recommendations. It also can make more detailed information available, as requested by the user.

ASSIST was also designed to be flexible and expandable. In particular, it is expected that the evaluation framework developed in this research will be expanded and modified. Hence, the form of the evaluation framework has been designed into the knowledge base, and the framework itself is completely separate from all other

elements of ASSIST. This means that the features and criteria which constitute the framework may be changed in the knowledge base at any time, in much the same manner as data is changed. This would have absolutely no impact on the rest of the system as long as the structure of the knowledge does not change. The framework has two main components, features and criteria, and each is structured hierarchically.

It is possible that the calculations used in the decision analysis will also require modification based on user experience with ASSIST. In particular, the calculations used for possibly dependent top level criteria should be examined carefully, as they may need to be changed if they are not producing the desired effect. Therefore, these calculations are performed by algorithms which are also isolated in the ASSIST knowledge base. Again, a change would have no effect on the rest of ASSIST as long as the structure of the knowledge does not change.

A prototype of ASSIST has been developed as a part of the design effort. This prototype has not only helped to illuminate important areas in the design specification, but it has also demonstrated the viability of the design. In addition, use of the prototype has made it clear that the basic ASSIST design is valid for all types of software. Only a few changes in the way the user specifies the type of software to be selected would be required to make ASSIST applicable to any type of software, rather than just to APSEs and their components. The knowledge base would not need to be changed at all. Although some embellishments may be desirable,

these would not require a change of structure, so they are provided for in the current ASSIST design.

Of course, the ASSIST prototype is not a complete implementation of the design, nor is it particularly sophisticated. However, the most important features of the design have been incorporated. All of the evaluation framework which has been designed is used in the prototype. The main restrictions are that the prototype only works with compilation systems, and it only uses one level of detail, both in specifying the important features and criteria to be used in arriving at the recommendations, as well as in presenting the recommendations.

The prototype was particularly helpful in specifying user *interface requirements* for ASSIST. Although the user interface of the prototype lacks sophistication, it was developed sufficiently to demonstrate the importance of a system which is relatively simple and easy to use. The utility of the support functions is clear, as is the importance of using many of the concepts of hypertext, such as chunking and non-linearity, in presenting information. Such requirements were levied upon the ASSIST design.

Future Directions

Although this research has made significant progress in advancing software evaluation technology, it has only provided a basic framework for future work. The data acquisition process must be carefully considered in completing the design of the Knowledge Acquisition Subsystem of ASSIST, and a plan must be devised for

collecting data for the database. Furthermore, along with the development of a full ASSIST implementation, the contents of the evaluation framework should be made more complete and the calculations used in the decision analysis should be examined closely. Finally, an implementor must determine the most effective mode for making ASSIST available to the potential user. Each of these needs will be considered in turn.

This research has not fully developed the Knowledge Acquisition Subsystem of the ASSIST design. Although there are no new concepts involved in designing this subsystem, it will require thoughtful consideration because it is imperative that it be as flexible as possible. Acquiring data for the database will not be particularly easy, especially at first, so the system must be able to deal with data in many different forms. It would be useful to extend the knowledge base to provide for these various forms, as well as the variety of terms which may be encountered. This would greatly enhance the system's ability to convert the data to the form and terminology required, while minimizing the effort needed to prepare the data for input.

Given the reluctance encountered in trying to collect a small amount of data to use in the ASSIST prototype, data collection may become the biggest roadblock to overcome before ASSIST can be put to its intended use. There are two major approaches which an organization can use to collect evaluation data. Either is expensive, and it is probably most effective for most organizations to use a combination of the two.

Evaluations can be done in-house, or outside organizations can be paid to provide data on assessments they have done. The advantage to providing in-house evaluations is that an organization can control the quality of the process. This can ensure "good" data, but only if the organization is willing to dedicate the time and resources to do the job well enough. Necessary resources include technical people, computer facilities, and administrative support. Just the computer hardware and software required to perform evaluations on some variety of systems, enough to establish a minimally useful database, would be more of an expense than all but the largest of organizations are usually willing to incur. If a commitment were made to provide such systems, along with the required people and expertise to use them effectively for performing evaluations, an organization would be set up as a software evaluation center. Hence, such an organization would probably be willing to sell the assessment data acquired in an attempt to justify the investment and offset its cost.

This brings us to the second alternative of purchasing assessment data. Even if an organization chooses to do some of its own assessments, unless it becomes an evaluation center, it will need to purchase a considerable amount of additional data before it will have enough on which to base intelligent software selection decisions.

There are two major difficulties to overcome before the concept of a software evaluation center catches on in the software community. First, decision makers must begin to recognize the scope

and amount of data required to make an intelligent decision. Second, an organization choosing to become an evaluation center must also recognize the necessary scope of data required, and it must be willing and able to provide all, or at least a significant portion, of it for each software product evaluated. Both of these difficulties can be overcome once a full evaluation framework, such as that provided in ASSIST, is recognized by those involved in the software assessment process.

However, it is likely that evaluation centers providing only a small portion of the overall assessment picture, such as is the current operating concept of the AES of the MOD [123], [124], [162] and the proposed DoD Ada evaluation center [41], are doomed to failure. Their products will, of necessity, be expensive, but the scope will be so limited as to make the expense unjustifiable. Unfortunately, the failure of these centers may perpetuate the current position many have taken, that complete evaluations cannot be cost-effective. Especially for selection decisions which must be made for very large systems, such as those used in DoD weapons systems, there is no basis for this conclusion. It is just that no evaluation of an Ada compilation system which is anywhere near complete has ever been made public. Mostly, this is because until now there has been no complete framework which could consider all the various types of assessment information which are potentially important. ASSIST provides such a framework. Given this framework, there is no longer a reason that relatively complete sets of data cannot be collected.

The ASSIST framework can be used most effectively in a computer system such as is specified in the ASSIST design. Decision theory is based on the premise that an organization must use a decision model when large amounts of data are involved in making a decision. ASSIST provides such a model in the form of a DSS which "understands" the framework of the factors involved in the decision process.

In order to use ASSIST effectively, it must be implemented according to the requirements and design specified in this research. Unfinished parts must also be completed. This includes the Knowledge Acquisition Subsystem, discussed earlier, as well as the knowledge structure. The features and criteria developed in this research must be expanded and validated, based on use. The calculations used to perform the decision analyses must also be validated and modified if necessary. The effects of criteria dependencies on the calculations must be examined. The prototype will be a valuable tool in the validation process. However, validation should be an ongoing process used with full ASSIST implementations as well. Expansion of the features should include both details which apply generally to any type of software, as well as details which are specific to one type or family of types. Expansion of criteria details will be more general and probably also more limited. It remains to be seen how many levels of detail will be necessary or optimal in feature or criteria specification.

ASSIST implementors must ultimately determine the target mode of use for the system. Either the implementor could take on

the task of collecting the data for the database, or the system could be leased or sold with an empty or minimal database. If the implementor provides the database, the system will be ready to use on delivery. However, the implementor then has the major task of keeping the data current. To be effective, updates would be necessary multiple times a year, and the delay time between data collection and updated database distribution must be minimal. However, if the using organization must provide its own data, demand would be lower for the product because only large organizations could probably afford to populate the database. The using organization would also have to possess considerably more expertise so the system could be properly prepared for use.

Again, a combination of the two approaches would probably be most effective. An implementor could provide current data on smaller systems and those most likely to interest smaller organizations (or possibly even individuals). Larger organizations could then supplement the database with more specialized data either gathered in-house or purchased from an evaluation center.

Summary

This research has resulted in the design for a DSS which can provide valuable assistance to the decision maker faced with selecting an Ada environment or some of its components. A variation of the design could also be suitable for use when selecting any type of software in general. A DSS such as ASSIST does not yet exist in a full implementation, but it is sorely needed. At present,

most software selection decisions are based on data which covers only a very small portion of the scope of software assessments, and this amount of data is inadequate for making intelligent decisions. Only a computer system such as ASSIST can bring together an adequate amount of assessment data with a broad enough scope and be able to provide a decision maker with a consistent and comprehensive analysis of such data so that intelligent decisions are possible.

ASSIST has the potential for becoming a very important part of software evaluation technology. In addition to providing a design and a prototype for a capability to assist a decision maker in selecting software, it also provides an initial organizational framework on which expectations for future assessments can be based. Once the expectation is there, the market place will surely provide it.

References

- [1] R. J. Abbott, "Program design by informal English descriptions," *Communications of the ACM*, vol. 26, no. 11, November 1983.
- [2] Ada Runtime Environments Working Group, "A framework for describing Ada runtime environments," *Ada Letters*, vol. VIII, no. 3, May/June 1988.
- [3] D. R. Addleman, M. J. Davis, and P. E. Presson, *Specification Technology Guidebook*, RADC-TR-85-135, Rome Air Development Center, August 1985.
- [4] L. Adelman and M. L. Donnell, "Evaluating decision support systems: a general framework and case study," *Microcomputer Decision Support Systems*, S. J. Andriole, editor, Wellesley, MA: QED Information Sciences, 1986.
- [5] W. W. Agresti, "Framework for a flexible development process," *New Paradigms for Software Development*, W. W. Agresti, editor, Washington, DC: Computer Society Press, 1986.
- [6] W. W. Agresti, "The conventional software life-cycle model: its evolution and assumptions," *New Paradigms for Software Development*, W. W. Agresti, editor, Washington, DC: Computer Society Press, 1986.
- [7] W. W. Agresti, "What are the new paradigms?" *New Paradigms for Software Development*, W. W. Agresti, editor, Washington, DC: Computer Society Press, 1986.
- [8] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Reading, MA: Addison-Wesley, 1986.
- [9] R. M. Akscyn, D. L. McCracken, and E. A. Yoder, "KMS: a distributed hypermedia system for managing knowledge in organizations," *Communications of the ACM*, vol. 31, no. 7, July 1988.
- [10] ANSI/IEEE Standard 729-1983, *IEEE Standard Glossary of Software Engineering Terminology*, 1983.

- [11] ANSI/IEEE Standard 829-1983, *IEEE Standard for Software Test Documentation*, 1983.
- [12] ANSI/IEEE Standard 830-1984, *IEEE Guide to Software Requirements Specifications*, 1984.
- [13] Apple Computer, Inc., *HyperCard Script Language Guide: The HyperTalk Language*, Reading, MA: Addison-Wesley, 1988.
- [14] L. J. Arthur, *Measuring Programmer Productivity and Software Quality*, New York: John Wiley & Sons, 1985.
- [15] A. N. Badre , "Designing chunks for sequentially displayed information," *Directions in Human/Computer Interaction*, A. Badre and B. Shneiderman, editors, Norwood, NJ: Ablex Publishing Corp, 1982.
- [16] D. R. Barstow, H. E. Shrobe, and E. Sandewall, *Interactive Programming Environments*, Preface by the editors, New York: McGraw-Hill, 1984.
- [17] M. J. Bassman, G. A. Fisher, Jr., and A. Gargaro, "An approach for evaluating the performance efficiency of Ada compilers," *Ada Letters*, vol. V, issue 2, Special edition, Ada in Use, Proceedings of the Ada International Conference, September, October 1985.
- [18] G. D. Bergland, "A guided tour of program design methodologies," *Computer*, vol. 14, no. 10, October 1981.
- [19] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, May 1988.
- [20] B. W. Boehm, "Improving software productivity," *Computer*, vol. 20, no. 9, September 1987.
- [21] G. Booch, "Object-oriented development," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 2, February 1986.

- [22] G. Booch, *Software Components with Ada*, Reading, MA: Benjamin/Cummings, 1987.
- [23] G. Booch, *Software Engineering with Ada*, Second edition, Reading, MA: Benjamin/Cummings, 1987.
- [24] T. P. Bowen, G. B. Wigle, and J. T. Tsai, *Specification of Software Quality Attributes*, Volume II, AD A153989, Griffiss AFB, NY: Rome Air Development Center, February 1985.
- [25] M. D. Broido, "Technical correspondence," *Communications of the ACM*, vol. 30, no. 2, February 1987.
- [26] F. P. Brooks, Jr., "No silver bullet: essence and accidents of software engineering," *Computer*, vol. 20, no. 4, April 1987.
- [27] M. D. Brown, *Understanding PHIGS*, San Diego: Megatek Corporation, 1985.
- [28] V. Bush, "As we may think," *Atlantic Monthly*, vol. 176, no. 1, July 1945.
- [29] J. N. Buxton, *Requirements for Ada Programming Support Environments*, "STONEMAN", Department of Defense, February 1980.
- [30] G. D. Buzzard and T. N. Mudge, "Object-based computing and the Ada language," *Computer*, vol. 18, no. 3, March 1985.
- [31] D. N. Card, V. E. Church, and W. W. Agresti, "An empirical study of software design practices," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 2, February 1986.
- [32] P. A. Carlson, "Hypertext: a way of incorporating user feedback into online documentation," *Text, ConText, and HyperText*, E. Barrett, editor, Cambridge, MA: The MIT Press, 1988.
- [33] V. L. Castor, *Criteria for the Evaluation of ROLM Corporation's Ada Work Center*, Department of Defense, January 1983.

- [34] J. P. Cavano and F. S. LaMonica, "Quality assurance in future development environments," *IEEE Software*, vol. 4, no. 5, September 1987.
- [35] R. M. Clapp, L. Duchesneau, R. A. Volz, T. N. Mudge, and T. Schultze, "Toward real-time performance benchmarks for Ada," *Communications of the ACM*, vol. 29, no. 8, August 1986.
- [36] J. S. Collofello and J. J. Buck, "Software quality assurance for maintenance," *IEEE Software*, vol. 4, no. 5, September 1987.
- [37] Commission of European Community, *PCTE: A Basis for a Portable Common Tool Environment*, Functional Specifications, Fourth edition, 1986.
- [38] J. Conklin, "Hypertext: an introduction and survey," *Computer*, vol. 20, no. 9, September 1987.
- [39] J. Conklin, *A Survey of Hypertext*, MCC Technical Report Number STP-356-86, Rev. 2, Austin, TX: Microelectronics and Computer Technology Corp., December 1987.
- [40] B. J. Cox, *Object Oriented Programming*, Reading, MA: Addison-Wesley, 1986.
- [41] R. E. Crafts, editor, "Ada compiler formal evaluation to begin in summer of 1989," *Ada Strategies*, vol. 3, no. 4, Arlington, MA: Cutter Information Corp., April 1989.
- [42] S. Danforth and C. Tomlinson, "Type theories and object-oriented programming," *ACM Computing Surveys*, vol. 20, no. 1, March 1988.
- [43] M. J. Davis and D. R. Addleman, "A practical approach to specification technology selection," *Journal of Systems and Software*, vol. 6, 1986.
- [44] Department of Defense, *Ada Compiler Evaluation Capability (ACEC) Technical Operating Report Reader's Guide*, August 1988.

- [45] Department of Defense, *Ada Programming Language*, ANSI/MIL-STD-1815A, January 1983.
- [46] Department of Defense, *Common Ada Programming Support Environment (APSE) Interface Set (CAIS)*, MIL-STD-1838A, April 1989.
- [47] Department of Defense, *Defense System Software Development*, DoD-STD-2167A, February 1988.
- [48] Department of Defense, *E&V Classification Schema Report*, June 1987.
- [49] Department of Defense, *E&V Guidebook*, Version 1.1, August 1988.
- [50] Department of Defense, *E&V Reference Manual*, Version 1.1, October 1988.
- [51] Department of Defense, *Requirements for Evaluation and Validation of Ada Programming Support Environments*, Version 1.0, October 1984.
- [52] Department of Defense, *Requirements for Evaluation and Validation of Ada Programming Support Environments*, Version 2.0, November 1986.
- [53] M. P. Dolciani, W. Wooton, E. F. Beckenbach, and S. Sharron, *Modern School Mathematics: Algebra 2 and Trigonometry*, Boston: Houghton Mifflin, 1968.
- [54] D. R. Dolk and B. R. Konsynski, "Knowledge representation for model management systems," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 11, November 1984.
- [55] M. Dowson, "Integrated project support with IStar," *IEEE Software*, vol. 4, no. 6, November 1987.
- [56] D. W. Etherington, *Reasoning with Incomplete Information*, Los Altos, CA: Morgan Kaufmann Publishers, 1988.

- [57] A. Evans and K. J. Butler, editors, *DIANA Reference Manual*, Revision 3, DTIC AD-A128232, Tartan Laboratories, Inc., February 1983.
- [58] EVB Software Engineering, *Object-Oriented Design Handbook*, January 1985.
- [59] R. Firth, V. Mosley, R. Pethia, L. Roberts, W. Wood, *A Guide to the Classification and Assessment of Software Engineering Tools*, CMU/SEI-87-TR-10, Pittsburgh: Software Engineering Institute, August 1987.
- [60] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: the correct way to summarize benchmark results," *Communications of the ACM*, vol. 29, no. 3, March 1986.
- [61] J. Foreman and J. Goodenough, *Ada Adoption Handbook: A Program Manager's Guide*, Version 1.0, CMU/SEI-87-TR-9, Pittsburgh: Software Engineering Institute, May 1987.
- [62] R. S. Freedman, *Programming with APSE Software Tools*, Princeton, NJ: Petrocelli Books, 1985.
- [63] P. Freeman and A. I. Wasserman, *Software Development Methodologies and Ada* ("Methodman"), Department of Defense, November 1982.
- [64] S. French, *Decision Theory*, New York: John Wiley & Sons, 1986.
- [65] F. Gallo, R. Minot, and I. Thomas, "The object management system of PCTE as a software engineering database management system," *SIGPLAN Notices*, vol. 22, no. 1, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, January 1987.
- [66] J. D. Gannon, E. E. Katz, and V. R. Basili, "Metrics for Ada packages: an initial study," *Communications of the ACM*, vol. 29, no. 7, July 1986.

- [67] P. K. Garg, "Abstraction mechanisms in hypertext," *Communications of the ACM*, vol. 31, no. 7, July 1988.
- [68] A. Giacalone and S. A. Smolka, "Integrated environments for formally well-founded design and simulation of concurrent systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [69] J. Gilles and R. Ford, "A guided tour through a window oriented debugging environment for embedded real time Ada systems," *Proceedings of the Third International IEEE Conference on Ada Applications and Environments*, Washington, DC: Computer Society Press, 1988.
- [70] J. Goguen and M. Moriconi, "Formalization in programming environments," *Computer*, vol. 20, no. 11, November 1987.
- [71] A. Goldberg and D. Robson, *Smalltalk-80: The Language and Its Implementation*, Reading, MA: Addison-Wesley, 1983.
- [72] D. Goodman, *HyperCard Developer's Guide*, New York: Bantam Books, 1988.
- [73] F. G. Halasz, "Reflections on Notecards: seven issues for the next generation of hypermedia systems," *Communications of the ACM*, vol. 31, no. 7, July 1988.
- [74] K. R. Hammond, G. H. McClelland, and J. Mumpower, *Human Judgment and Decision Making*, New York: Praeger Publishers, 1980.
- [75] S. Harbaugh and J. A. Forakis, "Timing studies using a synthetic Whetstone benchmark," *Ada Letters*, vol. IV, no. 2, September, October 1984.
- [76] W. Harrison, "RPDE3: a framework for integrating tool fragments," *IEEE Software*, vol. 4, no. 6, November 1987.
- [77] D. R. Hofstadter, *Goedel, Escher, Bach*, New York: Vintage Books, 1979.

- [78] F. R. A. Hopgood, D. A. Duce, J. R. Gallop, and D. C. Sutcliffe, *Introduction to the Graphical Kernel System (GKS)*, Second edition, New York: Academic Press, 1986.
- [79] J. J. Horning, "Structuring compiler development," *Compiler Construction*, Second edition, New York: Springer-Verlag, 1976.
- [80] R. C. Houghton and D. R. Wallace, "Characteristics and functions of software engineering environments," *Software Engineering Notes*, vol. 12, no. 1, January 1987.
- [81] G. P. Huber, "The nature of organizational decision making and the design of decision support systems," *MIS Quarterly*, vol. 5, no. 2, June 1981.
- [82] S. E. Hudson and R. King, "The Cactis Project: database support for software environments," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [83] IEEE Standard 990-1986, *IEEE Recommended Practice for Ada As a Program Design Language*, 1986.
- [84] IEEE Standard 1016-1987, *IEEE Recommended Practice for Software Design Descriptions*, 1987.
- [85] K. E. Jordan, "Performance comparison of large-scale scientific computers: scalar mainframes, mainframes with integrated vector facilities, and supercomputers," *Computer*, vol. 20, no. 3, March 1987.
- [86] C. Kaehler, *HyperCard Power*, Reading, MA: Addison-Wesley, 1988.
- [87] K. Kandt, "On building future decision support systems," *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, vol. III, 1988.
- [88] A. Kay and A. Goldberg, "Personal dynamic media," *Computer*, vol. 10, no. 3, March 1977.

- [89] P. B. W. Keen and M. S. S. Morton, *Decision Support Systems: An Organizational Perspective*, Reading, MA: Addison-Wesley, 1978.
- [90] K. Kishida, T. Katayama, M. Matsuo, I. Miyamoto, K. Ochimizu, N. Saito, J. H. Saylor, K. Torii, and L. G. Williams, "SDA: a novel approach to software environment design and construction," *Proceedings of 10th International Conference on Software Engineering*, Singapore, April 1988.
- [91] B. G. Knapp, F. L. Moses, and L. H. Gellman, "Information highlighting on complex displays," *Directions in Human/Computer Interaction*, A. Badre and B. Shneiderman, editors, Norwood, NJ: Ablex Publishing Corp., 1982.
- [92] J. C. Knight and R. H. Crowe, "A system for evaluating Ada implementations using synthesized benchmarks," *Proceedings of the Third International IEEE Conference on Ada Applications and Environments*, Washington, DC: Computer Society Press, 1988.
- [93] T. Koschmann and M. W. Evens, "Bridging the gap between object-oriented and logic programming," *IEEE Software*, vol. 5, no. 4, July 1988.
- [94] J. E. Kottemann and W. E. Remus, "When and how cognitive style impacts decision making," *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, vol. III, 1988.
- [95] J. F. Kramer, P. Oberndorf, J. Long, C. Roby, R. M. Robinson, J. Clouse, and J. Chludzinski, *CAIS Reader's Guide for DoD-STD-1838*, Institute for Defense Analyses, August 1987.
- [96] L. Latour and E. Johnson, "Seer: a graphical retrieval system for reusable Ada software modules," *Proceedings of the Third International IEEE Conference on Ada Applications and Environments*, Washington, DC: Computer Society Press, 1988.

- [97] M. M. Lehman and W. M. Turski, "Essential properties of IPSEs," *Software Engineering Notes*, vol. 12, no. 1, January 1987.
- [98] P. E. Lehner, M. A. Probus, and M. L. Donnell, "Building decision aids: exploiting the synergy between decision analysis and artificial intelligence," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 4, July/August 1985.
- [99] P. M. Lewis II, D. J. Rosenkrantz, and R. E. Stearns, *Compiler Design Theory*, Reading, MA: Addison-Wesley, 1976.
- [100] T. E. Lindquist, "Assessing the usability of human-computer interfaces," *IEEE Software*, vol. 2, no. 1, January 1985.
- [101] T. E. Lindquist, P. K. Lawlis, and D. P. Levine, "Typing information in a software engineering environment," *Proceedings of the Sixth International Conference on Entity-Relationship Approach*, November 1987.
- [102] M. A. Linton, "Distributed management of a software database," *IEEE Software*, vol. 4, no. 6, November 1987.
- [103] Luqi, "Software evolution through rapid prototyping," *Computer*, vol. 22, no. 5, May 1989.
- [104] T. G. L. Lyons and J. C. D. Nissen, *Selecting an Ada Environment*, New York: Cambridge University Press, 1986.
- [105] J. Martin and S. Oxman, *Building Expert Systems*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [106] R. Martin, *Quality Factors for Operational Characteristics*, Personal communication, 1989.
- [107] J. McDermid and K. Ripken, *Life Cycle Support in the Ada Environment*, New York: Cambridge University Press, 1984.
- [108] G. McFarland, P. Brennan, J. D. Litke, and M. S. Restivo, "A tool set for distributed Ada programming," *Proceedings of the*

Third International IEEE Conference on Ada Applications and Environments, Washington, DC: Computer Society Press, 1988.

- [109] C. W. McKay, "A proposed framework for the tools and rules to support the life cycle of the space station program," *Proceedings of the IEEE Compass '87 Conference*, June 1987.
- [110] I. Miller and J. E. Freund, *Probability and Statistics for Engineers*, Second edition, Englewood Cliffs, NJ: Prentice-Hall, 1977.
- [111] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information," *Writings of the Revolution: Selected Readings on Software Engineering*, edited by E. Yourdon, New York: Yourdon Press, 1982.
- [112] R. P. Minch and J. R. Burns, "Conceptual design of decision support systems utilizing management science models," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 4, July/August 1983.
- [113] G. L. Mohnkern, "Choosing a PC Ada," *Computer Language*, vol. 5, no. 12, December 1988.
- [114] K. E. Nidiffer, "Implementation of standard software support environments," *Proceedings of AIAA Computers in Aerospace VI Conference*, October 1987.
- [115] J. C. D. Nissen, B. A. Wichmann, and others, "Guidelines for Ada compiler specification and selection," *Ada: Language, Compilers and Bibliography*, M. W. Rogers, editor, New York: Cambridge University Press, 1984.
- [116] D. S. Notkin and A. N. Habermann, "Software development environment issues as related to Ada," *Tutorial: Software Development Environments*, A. I. Wasserman, editor, New York: IEEE, 1981.

- [117] P. A. Oberndorf, "The Common Ada Programming Support Environment (APSE) Interface Set (CAIS)," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [118] T. L. Pappas, "Ada on the IBM PC -- it's for real!," *Computer*, vol. 21, no. 9, September 1988.
- [119] D. L. Parnas, "Designing software for ease of extension and contraction," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 3, March 1979.
- [120] M. H. Penedo and W. E. Riddle, "Guest editors' introduction: software engineering environment architectures," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [121] D. E. Perry and G. E. Kaiser, "Models of software development environments," *Proceedings of 10th International Conference on Software Engineering*, Singapore, April 1988.
- [122] N. Petreley, "Relational databases," *InfoWorld*, vol. 10, issue 16, April 18, 1988.
- [123] R. H. Pierce, "Evaluating Ada implementations -- smoothing the transition to Ada," *Ada: Managing the Transition*, Proceedings of the Ada-Europe International Conference, Edinburgh, P. J. L. Wallis, editor, London: Cambridge University Press, 1986.
- [124] R. H. Pierce, I. Marshall, and S. D. Bluck, *An Introduction to the MOD Ada Evaluation System*, Software Sciences Ltd., 1986.
- [125] C. W. Pittman, "The space station information system and software support environment," *Proceedings of 10th International Conference on Software Engineering*, Singapore, April 1988.
- [126] R. S. Pressman, *Making Software Engineering Happen*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [127] R. S. Pressman, *Software Engineering*, Second edition, New York: McGraw-Hill, 1987.

- [128] J. Price, "Creating a style for online help," *Text, ConText, and HyperText*, E. Barrett, editor, Cambridge, MA: The MIT Press, 1988.
- [129] S. R. Rainier, T. P. Reagan, and A. E. Salwin, "The benchmark generator tool: measuring Ada compilation system performance," *Proceedings of the Third International IEEE Conference on Ada Applications and Environments*, Washington, DC: Computer Society Press, 1988.
- [130] S. P. Reiss, "Working in the Garden environment for conceptual programming," *IEEE Software*, vol. 4, no. 6, November 1987.
- [131] P. Rivett, *Model Building for Decision Analysis*, New York: John Wiley & Sons, 1980.
- [132] S. Rosen, *Lectures on the Measurement and Evaluation of the Performance of Computing Systems*, Philadelphia: Society for Industrial and Applied Mathematics, 1976.
- [133] W. B. Rouse, "Design and evaluation of computer-based decision support systems," *Microcomputer Decision Support Systems*, S. J. Andriole, editor, Wellesley, MA: QED Information Sciences, 1986.
- [134] A. P. Sage, "An overview of contemporary issues in the design and development of microcomputer decision support systems," *Microcomputer Decision Support Systems*, S. J. Andriole, editor, Wellesley, MA: QED Information Sciences, 1986.
- [135] A. P. Sage, "An overview of system design for human interaction," *System Design for Human Interaction*, A. P. Sage, editor, New York: IEEE Press, 1987.
- [136] A. P. Sage, "Behavioral and organizational considerations in the design of information systems and processes for planning and decision support," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-11, no. 9, September 1981.

- [137] B. Shneiderman, *Designing the User Interface*, Reading, MA: Addison-Wesley, 1987.
- [138] D. P. Shoemaker, C. W. Garland, and J. I. Steinfeld, *Experiments in Physical Chemistry*, New York: McGraw-Hill, 1974.
- [139] M. S. Silver, "User perceptions of DSS restrictiveness: an experiment," *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, vol. III, 1988.
- [140] J. B. Smith and S. F. Weiss, "An overview of hypertext," *Communications of the ACM*, vol. 31, no. 7, July 1988.
- [141] G. W. Snedecor and W. G. Cochran, *Statistical Methods*, Sixth edition, Ames, IA: The Iowa State University Press, 1967.
- [142] R. A. Snowdon, N. C. Munro, N. W. Davis, and M. I. Jackson, "Advanced support environments: an industry viewpoint," *Integrated Project Support Environments*, London: Peter Peregrinus Ltd., 1985.
- [143] W. E. Souder, *Management Decision Methods*, New York: Van Nostrand Reinhold, 1980.
- [144] R. H. Sprague, Jr., "A framework for the development of decision support systems," *MIS Quarterly*, vol. 4, no. 4, December 1980.
- [145] V. Stenning, T. Froggatt, R. Gilbert, and E. Thomas, "The Ada environment: a perspective," *Computer*, vol. 14, no. 6, June 1981.
- [146] H. G. Stuebing, "A software engineering environment (SEE) for weapon system software," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 4, July 1984.
- [147] M. M. Tanik and R. T. Yeh, "Guest editors' introduction: Rapid prototyping in software development," *Computer*, vol. 22, no. 5, May 1989.

- [148] R. N. Taylor and T. A. Standish, "Steps to an advanced Ada programming environment," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 3, March 1985.
- [149] A. Tetewsky and R. Racine, "Ada compiler selection for embedded targets," *Ada Letters*, vol. VII, no. 5, September, October 1987.
- [150] Texas Instruments, *Ada Compiler Benchmark Report*, Version 1.0, December 1987.
- [151] M. Todd, "Language offers sophisticated programming environment," *InfoWorld*, vol. 10, issue 9, February 29, 1988.
- [152] A. van Dam, "Hypertext '87: keynote address," *Communications of the ACM*, vol. 31, no. 7, July 1988.
- [153] A. van Lamsweerde, B. Delcourt, E. Delor, M. C. Schayes, and R. Champagne, "Generic lifecycle support in the ALMA environment," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [154] D. W. Ver Planck and B. R. Teare, Jr., *Engineering Analysis: An Introduction to Professional Method*, New York: John Wiley & Sons, 1954.
- [155] M. S.-Y. Wang and J. F. Courtney, Jr., "A conceptual architecture for generalized decision support system software," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-14, no. 5, September/October 1984.
- [156] A. I. Wasserman and C. J. Prenner, "Toward a unified view of data base management, programming languages, and operating systems -- a tutorial," *Information Systems*, vol. 4, no. 2, February 1979.
- [157] A. I. Wasserman, "Software tools in the user software engineering environment," *Tutorial: Software Development Environments*, A. I. Wasserman, editor, New York: IEEE, 1981.

- [158] A. I. Wasserman, "Toward integrated software development environments," *Tutorial: Software Development Environments*, A. I. Wasserman, editor, New York: IEEE, 1981.
- [159] D. A. Waterman, *A Guide to Expert Systems*, Reading, MA: Addison-Wesley, 1986.
- [160] N. Weiderman, A. N. Habermann, M. W. Borger, and M. H. Klein, "A methodology for evaluating environments," *SIGPLAN Notices*, vol. 22, no. 1, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, January 1987.
- [161] N. Weiderman, *Ada Adoption Handbook: Compiler Evaluation and Selection*, Version 1.0, AD A207717, Software Engineering Institute, March 1989.
- [162] N. Weiderman, "The Ada Evaluation System," Presentation to the E&V Team, March 1988.
- [163] S. A. Weyer, "As we may learn," *Interactive Multimedia*, S. Ambron and K. Hooper, editors, Redmond, WA: Microsoft Press, 1988.
- [164] T. Wheeler, "An example of the developer's documentation for an embedded computer system written in Ada," *Ada Letters*, vol. VI, no. 6 and vol. VII, no. 1, November, December 1986 and January, February 1987.
- [165] D. J. White, *Fundamentals of Decision Theory*, New York: American Elsevier, 1976.
- [166] R. Wiener and R. Sincovec, *Software Engineering with Modula-2 and Ada*, New York: John Wiley & Sons, 1984.
- [167] T. Winograd, "Beyond programming languages," *Interactive Programming Environments*, D. R. Barstow, H. E. Shrobe, and E. Sandewall, editors, New York: McGraw-Hill, 1984.

- [168] A. L. Wolf, L. A. Clarke, and J. C. Wileden, "Ada-based support for programming-in-the-large," *IEEE Software*, vol. 2, no. 2, March 1985.
- [169] N. Yankelovich, B. J. Haan, N. K. Meyrowitz, and S. M. Drucker, "Intermedia: the concept and the construction of a seamless information environment," *Computer*, vol. 21, no. 1, January 1988.
- [170] G. Younggren, "Using an object-oriented programming language to create audience-driven hypermedia environments," *Text, ConText, and HyperText*, E. Barrett, editor, Cambridge, MA: The MIT Press, 1988.
- [171] W. M. Zage, H. E. Dunsmore, D. M. Zage, G. Cabral, *A Tool for Evaluating Software Engineering Environments*, SERC-TR-2-P, West Lafayette, IN: Software Engineering Research Center, Purdue University, 1988.
- [172] M. Zeleny, *Multiple Criteria Decision Making*, New York: McGraw-Hill, 1982.
- [173] I. Zualkernan, W. T. Tsai, and D. Volovik, "Expert systems and software engineering: ready for marriage?" *IEEE Expert*, vol. 1, no. 4, Winter 1986.

**SUPPORTING SELECTION DECISIONS
BASED ON THE TECHNICAL EVALUATION OF
ADA ENVIRONMENTS AND THEIR COMPONENTS**

by

Patricia K. Lawlis

VOL. II

**A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy**

ARIZONA STATE UNIVERSITY

December 1989

Appendix A

Software Evaluation Questionnaire

Software Evaluation Questionnaire

This questionnaire will be used to help determine the requirements for a decision support system to be developed as a part of Pat Lawlis' PhD dissertation research. The purpose of the system is to assist a decision maker in selecting which Ada Programming Support Environment (APSE) and/or tools to purchase in support of an upcoming development project. The system will be able to use evaluation information which has been gathered from numerous sources.

All responses will be kept confidential. Please return this questionnaire within a week of receipt, or by the end of this month (November 1988) at the very latest. Your support is appreciated, and if you so indicate in question 4, you will have the opportunity to get a demonstration of the next system prototype when it is ready.

Please return the questionnaire to: Major Pat Lawlis
3318 E. Dry Creek Rd.
Phoenix, AZ 85044

1. What type of information would you like to see such a system provide? (Rank order all that apply)

- _____ A recommendation of one "best" choice of APSE or specific tool
- _____ A "graded" list of all APSEs and/or tools which meet user-given requirements
- _____ Summaries of available evaluation information
- _____ Raw evaluation data
- _____ Explanations for given evaluation "ratings"
- _____ Information on possible evaluation information which is missing (not available)
- _____ Other (please explain)

2. What criteria should the system user provide (or perhaps select) to establish the context for which an APSE or tool should be evaluated? (Rank order all that apply)

- _____ Requirements which absolutely must be met
How should such requirements be given by the user?
- _____ The application area (real-time, database, etc.)
List areas which you think should be considered:

- _____ Life cycle activities of interest
List the activities you think should be considered:

 - _____ Tool sets of interest (compilation system, program management, documentation system, etc.)
List tools sets you think should be considered:

 - _____ Attributes of interest (usability, maintainability, portability, etc.)
List attributes you think should be considered:

 - _____ Functions of interest (formatting, translation, analysis, etc.)
List functions you think should be considered:

 - _____ The sources of the evaluation information to be considered by the system (vendor supplied, test suite information, user feedback, etc.)
List sources you think should be considered:

 - _____ Weights for selected criteria to indicate their relative importance
What is the maximum number of criteria which should be weighted?
List the criteria you think should be considered:

 - _____ Other (please explain)
-
3. Would you use such a system if it were available?
If no, why not?

 4. Would you like a demonstration of the next prototype version of the system?
If yes, please give name, address, and phone number.

 5. Please add any other comments or suggestions which you think are pertinent. Use the back or additional sheets if necessary.

Appendix B

Notes on the ASSIST Prototype Version 1.0

Notes on the ASSIST Prototype Version 1.0

Please comment on any or all of the following as you are working with the ASSIST prototype. Thank you for your valued feedback.

1. Does the system establish all the necessary characteristics for the software to be selected (type of software, application area, features, criteria, etc.), or does it have the appropriate "hooks" to be able to do so in the future? If not, what should be added, deleted, or changed?
2. Is the weighting process used appropriately? If not, please suggest appropriate changes.
3. Does the system provide the right type of information to help a decision maker select software, or does it have the appropriate "hooks" to provide it in the future? If not, what has not been accounted for? Please be as specific as possible.
4. Do you find the system easy to use? Please comment on the user interface, the User Manual, and/or the on-line Help.
5. Please add any other comments or suggestions which you think are pertinent. Use the back if necessary.

Appendix C
Requirements for ASSIST

1. Introduction

This section provides an overview of a decision support system (DSS) called the Ada Software Selection assISTant (ASSIST). ASSIST's purpose is to support decisions on the selection of Ada environments and their components, where the selection is based on the technical evaluation of the software.

1.1 Purpose

The purpose of this software requirements specification is to specify the requirements for ASSIST. The intended audience for this document consists of software engineers, other software professionals, and technical managers.

1.2 Scope

ASSIST will provide assistance to a technical manager who must make decisions concerning which software should be selected to support the development of a particular Ada software system. The specification and design of ASSIST is a research effort to prove that software selection is a viable concept for a decision support system. This effort includes prototyping the software as a part of developing and refining the requirements in this document.

ASSIST shall provide the following for the decision maker (also referred to as the user):

- 1) the interactive choice of features and criteria to use in the software selection, with multiple levels of detail available in the descriptions of the features and criteria
- 2) the interactive selection of the level of detail provided in the selection recommendations and other information given by the system, including data and statistical summaries where appropriate

- 3) a user interface which is easy to understand and easy to use, with easy access to on-line help and no system expertise assumed
- 4) a mechanism for keeping track of the "current" reasoning process, including the selections which have been made by the user and the choices which are now available, with the ability to view this at any time
- 5) the interactive selection of hard copy reports, including printing the current window or printing the information provided in a specific field in the window
- 6) a browsing mechanism which permits the user to "see" the organization of the selection process and also permits the user to move to another place in the process, if desired
- 7) a mechanism for saving the features and criteria selected in a program session and retrieving these for use in a subsequent session

In addition, the system must provide a mechanism for a system manager to access the system and be able to do the following:

- 8) add to or update evaluation data in the system database
- 9) add to or update the feature and criteria knowledge in the system knowledge base

The following software products, typical of a DSS, have been chosen for ASSIST's architecture:

A. User Interface Subsystem

This subsystem provides the resources for developing the screen windows and reports and for capturing user input. This subsystem does not provide the logic for directing system processing.

B. Knowledge Acquisition Subsystem

This subsystem handles the acquisition of evaluation data for the database and the acquisition of criteria knowledge for the knowledge base. It accepts data or knowledge input via the User Interface

Subsystem in any number of predefined formats, and it provides for the conversion of the input into the appropriate form for entry into the system database and knowledge base. It accesses the Knowledge Base Subsystem to store the input into the database or knowledge base.

C. Decision Logic Subsystem

This subsystem provides the main decision logic of the system. It provides appropriate windows for either collecting selection information from the user or providing information to the user. It accepts interactive input from the user via the User Interface Subsystem. It leads the user through the decision process, responding to user actions, and accessing the Knowledge Base Subsystem as necessary.

D. Knowledge Base Subsystem

This subsystem provides access to both the database and the knowledge base of the system. It is the only mechanism for storing and retrieving system information.

1.3 Definitions, Acronyms, and Abbreviations

APSE - Ada Programming Support Environment - the name given to a software engineering environment designed to support the full life cycle of both technical and management activities involved in developing a software system in Ada. Because environments with such complete support do not yet exist, the name is also currently used for any set of tools which supports development in Ada.

chunk - a logical grouping of information, typically text, which embodies a single concept or idea and is small enough to be viewed at one time.

DSS - Decision Support System - the term for a software system which supports a decision maker in arriving at an important decision. This term is typically used for a system supporting a process which is not well structured and not well understood. Such a system does not necessarily come up with single answers for the decision maker.

This is in contrast with an expert system which typically supports a well understood process and provides a single answer to the problem in question.

evaluation data - the raw evaluation data which comes from sources outside this system (for example, the statistical data resulting from the evaluation of a compiler)

evaluation criteria - characteristics by which the goodness of the software is measured (for example, the usability of the software)

evaluation features - characteristics which are either present or absent in the software, and no measure of goodness is associated with them (for example, a particular hardware/operating system configuration required for running the software)

hypertext - a nonlinear method for organizing textual chunks

weighting factor - a number associated with the importance of either an evaluation criteria or feature, where the larger the number, the more important the criteria or feature in the software selection process

window - information viewed together on the screen and usually enclosed in a box, possibly including both text and graphics, which comprises a complete chunk of information. It may or may not actually fill the entire screen.

1.4 References

[1] P. K. Lawlis, Supporting Selection Decisions Based on the Technical Evaluation of Ada Environments and Their Components, PhD Dissertation Proposal, Arizona State University, 28 October 1988. (The bibliography of this document contains extensive additional references.)

[2] P. K. Lawlis, Test Plan for Ada Software Selection assISTant (ASSIST), produced as part of PhD dissertation work, Arizona State University, 9 March 1989.

1.5 Overview

Section 2 provides a perspective on the requirements for ASSIST, and then Section 3 provides the specific system requirements.

2. General Description

The given requirements were produced as a part of the academic research for a PhD dissertation. As a part of the research, a system prototype was developed. The evaluation of the prototype provided feedback to this document.

2.1 Product Perspective

This is a stand alone product. It is intended to be run on contemporary engineering workstation.

2.2 Product Functions

Input - the system will have the ability to input evaluation information into its database and knowledge base. It will also provide for interactive input of selections from the user.

Processing - The system will determine the nature of the software to be selected, and then it will employ user-chosen evaluation features and criteria to determine the recommendations to be provided to the decision maker user. The user will be able to select from various features and criteria at multiple levels of detail, and to provide relative weighting factors for each selected feature and criterion. The system will provide consistency in the application of these selections to the information in the database, using the knowledge in the knowledge base to determine how to apply them.

Output - Recommendations and other information will be provided to the user interactively and at multiple levels of detail, with the option that it may also be printed.

2.3 User Characteristics

The regular system user will be an occasional user who will be a decision maker, typically a technical manager. The system manager

will set up the database and the knowledge base for the decision maker to use. To distinguish between the two in this document, the term "user" will always refer to the decision maker.

2.4 General Constraints

This system has been constrained to supporting decisions on the selection of Ada environments and their components. The implementation of the Decision Logic Subsystem will be affected by this constraint, but all of the other subsystems, and all of the subsystem interfaces, will be appropriate for any software selection process in general.

2.5 Assumptions and Dependencies

The hardware available for the system is assumed to be a computer system which can provide adequate disk storage space for as much evaluation data as is available, acceptable execution speed, and a user interface which includes graphics capabilities as well as both keyboard and mouse input. The software will be implemented either in Ada or some higher level object oriented language. Adequate software tools which can support an interactive user interface using both text and graphics is assumed.

3. Specific Requirements

This section details the individual, specific software system requirements. Each of these requirements is cross-referenced to other sections of this document. Cross-referenced sections are given in brackets at the beginning of each requirement statement. Each requirement may be verified by inspection during testing unless another method is specified.

3.1 Functional Requirements

3.1.A The following requirements apply to the process of entering evaluation information into the database:

3.1.A.1 [1.2.8, 1.2.B, 2.2, 2.3] Evaluation data collected from outside sources can be entered into the system database. This data can be accepted either interactively or from a file. For data accepted from a file, the system will be able to accept evaluation data from at least two different file formats, at least one of which will be based on the use of keywords in the data file. The system must convert the input into the proper form for the database.

3.1.A.2 [1.2.8, 1.2.B, 1.2.D, 2.2] Evaluation data already in the database may be changed or deleted interactively.

3.1.A.3 [1.2.8, 1.2.A, 1.2.B, 1.2.D, 2.2] All input to the database will be processed by the Knowledge Acquisition Subsystem. Interactive entries will be input via the User Interface Subsystem. The database will be accessed only via the Knowledge Base Subsystem.

Verification of this requirement requires code inspection in addition to regular testing procedures. It must be verified that the only mechanism for input to the database is provided in the Knowledge Acquisition Subsystem, and it must also be verified that the only access to the database comes through the Knowledge Base Subsystem.

3.1.A.4 [1.2.D] The Knowledge Base Subsystem will provide a mechanism for consistency checking among the entries in its database.

3.1.B The following requirements apply to the process of entering information into the knowledge base:

3.1.B.1 [1.2.9, 1.2.B, 1.2.D, 2.2] Knowledge of evaluation features and criteria in the system knowledge base can be added to, changed, or deleted interactively.

3.1.B.2 [1.2.9, 1.2.A, 1.2.B, 1.2.D, 2.2] All interactive input to the knowledge base will be processed by the Knowledge Acquisition Subsystem via the User Interface Subsystem, and the knowledge base will be accessed only via the Knowledge Base Subsystem.

Verification of this requirement requires code inspection in addition to regular testing procedures. It must be verified that the only mechanism for input to the knowledge base is provided in the Knowledge Acquisition Subsystem, and it must also be verified that the only access to the knowledge base comes through the Knowledge Base Subsystem.

3.1.B.3 [1.2.D] The Knowledge Base Subsystem will provide a mechanism for consistency checking among the entries in its knowledge base.

3.1.C The following requirements apply to interactive input by the decision maker:

3.1.C.1 [1.2.1, 1.2.7, 1.2.C, 2.2] The user feature and criteria selections can be entered either interactively or from a file saved from a previous session.

3.1.C.2 [1.2.1, 2.2] If entering feature and criteria selections interactively, the user will have the ability to make changes to any part of the selections at any time, rather than being restricted to entering selections in only one particular order.

3.1.C.3 [1.2.1, 1.2.7, 2.2] After entering feature and criteria selections from a file, the user will be able to make interactive modifications to these selections just as if they had been entered interactively in the first place.

3.1.C.4 [1.2.1, 1.2.3, 1.2.5, 1.2.8, 1.2.9, 2.2] Input will be verified by the system as it is entered. If the input is not appropriate, it will be discarded and new input will be requested. Appropriate informative error messages will be provided.

3.1.D The following requirements apply to system outputs:

3.1.D.1 [1.2.7, 2.2] The selection features and criteria can be saved to a file at any time during a session by a single user action. This file can then be used in a subsequent session.

3.1.D.2 [1.2.5, 2.2] At any time during a session, the user can cause the information in the current window to be sent to the printer. An option will be available to the user to print the information exactly as it looks in the window or else to print a report which clearly provides all or some portion of the information which is displayed in the window.

3.1.D.3 [1.2.2, 2.2] After the user has completely specified all required selection features and criteria, the system will provide the user with windows of evaluation recommendations, information, and analysis at multiple levels of detail. The first (highest) level of detail will provide recommendations to the user, while the second level of detail will provide analysis information. This analysis information will include an explanation of calculations used to arrive at the recommendations, and it will provide the user with the ability to change parameters used in these calculations. Further levels will provide progressively more detailed information until the lowest level will provide the user with the evaluation data from the database.

3.1.E The following requirements apply to system processing:

3.1.E.1 [1.2.3] Any icons or other conventions used will be clearly explained to the user at the beginning of a session. Optionally, the information will be available in a help window, via one mouse click or one keystroke, to the user who wants to see such explanations. In any case, the display of the help window containing such explanations will be an option available to the user from any window.

3.1.E.2 [1.2.3] The active window will clearly invite one or more of the following responses from the user:

- (a) One or more choices, as appropriate, may be selected by the click of a mouse (or equivalent) in a clearly marked area of the screen which is associated with a (clear) description of the choice.
- (b) An action may be selected by the click of a mouse (or equivalent) in a clearly marked area of the screen which is associated with a (clear) description of the action.
- (c) Text may be input from the keyboard into a field which is highlighted on the screen and/or where the cursor is located.

3.1.E.3 [1.2.1, 1.2.3, 1.2.6, 1.2.C, 2.2] The system will clearly lead the decision maker through the process of selecting features and criteria in preparation for the summary of recommendations and information to be provided to the decision maker. This process must be complete before the summary can be provided.

3.1.E.4 [1.2.6, 2.2] The system will determine from the decision maker the type of software about which to make recommendations. The software type may be specified as a particular individual tool, a particular tool set, the software which will support a particular life cycle activity, or an entire APSE.

3.1.E.5 [1.2.6, 2.2] The application area for the software will be specified by the decision maker. This information will be used to assist the decision maker by suggesting features and criteria which are usually important in making software selections for the particular application area.

3.1.E.6 [1.2.1, 1.2.C, 1.2.D, 2.2] The system will provide the decision maker with suggested features for selection. Other features may be added by the user, but they must be consistent with the features in the knowledge base. The system will also provide the capability for using multiple levels of detail for feature specification. At the finest level of detail for any given feature, the user will be able to specify that feature to the finest level of detail consistent with the knowledge in the system knowledge base for handling that feature. The system will provide default weights for all features selected, and the user will also be able to change those weights.

Verification of this requirement requires an inspection of the knowledge base in addition to regular testing procedures. It must be verified that all choices provided to the user are consistent with the knowledge in the knowledge base.

3.1.E.7 [1.2.1, 1.2.C, 1.2.D, 2.2] The system will provide the decision maker with suggested criteria for selection. Other criteria may be added by the user, but they must be consistent with the criteria in the knowledge base. The system will also provide the capability for using multiple levels of detail for criteria specification. At the finest level of detail for any given criterion, the user will be able to specify that criterion to the finest level of detail consistent with the knowledge in the system knowledge base for handling that criterion. The system will provide default weights for all criteria selected, and the user will also be able to change those weights.

Verification of this requirement requires an inspection of the knowledge base in addition to regular testing procedures. It must be verified that all choices provided to the user are consistent with the knowledge in the knowledge base.

3.1.E.8 [1.2.1, 1.2.3, 1.2.D, 2.2] When making selections, the decision maker will usually not be required to enter words using the keyboard. Choices will be provided which will require either a click of a mouse or one keystroke to select, except in cases where this is not practical. Selection choices will not be implemented so as to restrict the decision maker from having flexibility in making choices. However, it will ensure that selections always be valid and consistent with the current system knowledge base.

Verification of this requirement requires an inspection of the knowledge base in addition to regular testing procedures. It must be verified that the choices provided by the system are consistent with the knowledge in the knowledge base.

3.1.E.9 [1.2.2, 1.2.C, 1.2.D, 2.2] The system will provide a summary of recommendations and other information to the decision maker after the selection of features and criteria has been completed. All recommendations and information provided will be based on the evaluation information data in the database, and this data will be treated in accordance with the feature and criteria knowledge provided in the knowledge base.

Verification of this requirement requires an inspection of the database and the knowledge base in addition to regular testing procedures. It must be verified that the results provided by the system are consistent with the data in the database and the knowledge in the knowledge base.

3.1.E.10 [1.2.2, 1.2.C, 1.2.D, 2.2] The system will provide the decision maker with multiple levels of detail for viewing the summary of recommendations and other information. At the finest level of detail, the user will be able to review the actual data in the system database.

Verification of this requirement requires an inspection of the database in addition to regular testing procedures. It must be verified that the results provided by the system at the finest level of detail are the data in the database.

3.1.E.11 [1.2.6] A form of graphical browser will be available for the user to view at any time to maintain a perspective of the organization of the system processing, as well as how the current activity fits into this processing.

3.1.E.12 [1.2.4, 2.2] At any time during a session, the user can cause the system to display a summary of the current selections and/or the reasoning used to get the summary of recommendations and information resulting from the current selections.

3.1.E.13 [1.2.3] During periods of delay, when the system is processing before putting a new display on the screen, the system will always provide the user with a clear indication that the system is working.

3.1.E.14 [1.2.3, 1.2.C] The user will be able to quit from the system at any time. However, a reminder will always be given that the user may want to save the current selections first (if that has not already been done).

3.1.E.15 [1.2.3] At any time that a user request cannot be satisfied or an error condition occurs, an appropriate informative message will be provided to the user, and regular program execution will continue.

3.1.E.16 [1.2.3] On-line help will always be available, with the click of a mouse or one keystroke, to give more complete explanations for the options possible from the current window.

3.2 Performance Requirements

All times given in this section refer to "wall clock" time, which can be verified with a stop watch.

3.2.1 [1.2.3, 2.2] The system must have a response time of less than 5 seconds from the time of any interactive input causing a new window to be displayed until that new window is displayed on the screen.

3.2.2 [1.2.3] A window must be displayed in its entirety, from first pixel to last, in less than 2 seconds.

3.2.3 [1.2.3] The average response time for each interactive input, except for the processing which displays summary recommendations, will be less than 5 seconds.

3.2.4 [1.2.3] The full response time for the processing which displays summary recommendations will be expected to vary greatly depending upon the amount of data in the database. However, the maximum delay time between displays of at least some portion of the summary output will be 30 seconds, and the total delay time will

not exceed 1 second for every evaluation in the database which is processed for the type of software specified by the user.

3.2.5 [1.2.8, 1.2.9, 1.2.D] The size of the database and knowledge base used by the system will only be restricted by the size of the storage area available. It will not be artificially restricted by the system.

Verification of this requirement requires code inspection in addition to regular testing procedures. It must be verified that the code contains no artificial restrictions on the size of the database or the knowledge base.

3.3 Design Constraints

3.3.1 [1.2.C] The knowledge of what the system should do with the evaluation data in the database is contained in the knowledge base. The knowledge base contains the evaluation features and criteria to be used as well as the specifics of how they should be used. This means that changes to the knowledge base will change how the system will deal with the information in the database.

Verification of this requirement requires an inspection of the knowledge base in addition to regular testing procedures. It must be verified that the knowledge base contains the knowledge for how to use the evaluation features and criteria.

3.3.2 [1.2.C] The main system logic in the Decision Logic Subsystem will access the algorithms which use the knowledge in the knowledge base to determine how to deal with the data in the database. However, the logic will work based only on the form of the knowledge, not the content.

Verification of this requirement requires code inspection in addition to regular testing procedures. It must be verified that no code in the Decision Logic Subsystem relies on specific contents of the knowledge base nor has any knowledge information "hard coded".

3.3.3 [1.2.A, 1.2.B, 1.2.C, 1.2.D] Each of the subsystems will be independent of each other's implementation. Each will depend only on explicitly defined interfaces to use another's resources.

Verification of this requirement requires code inspection in addition to regular testing procedures. It must be verified that no code of any subsystem contains direct access to another subsystem domain without using the defined interfaces.

3.4 Attributes

3.4.1 [1.2.3, 1.2.8, 1.2.9, 1.2 .B, 2.2, 2.3] The expected system user is the decision maker. The system will provide a mechanism for a system manager to use the Knowledge Acquisition Subsystem to modify the database and/or the knowledge base, but this mechanism will not be visible to the decision maker.

3.4.2 [1.2.3] The system will always present an active window to the user. One window will not disappear without another immediately appearing.

3.4.3 [1.2.3] If other windows in addition to the active window are on the screen, the active window will always be clearly marked and fully visible.

3.4.4 [1.2.3] Screen windows will be able to provide graphics and/or text.

3.4.5 [1.2.3] The user will not be restricted to "backing out" from a particular level of processing (such as when viewing lower levels of detail) by retracing steps through every window that led there. The system must (at least appear to) employ the hypertext concept of nonlinear organization of chunks of information.

3.4.6 [2.1, 2.5] The system must run on a contemporary workstation which provides both graphics and textual capabilities.

Index

- Ada - 1, 1.2, 1.3, 1.4, 2.4, 2.5, 3.1.E.4
- application area - 3.1.E.5
- browser - 3.1.E.11
- consistency checking - 3.1.A.4, 3.1.B.3
- database - 1.2.7, 1.2.B, 1.2.D, 2.2, 2.3, 3.1.A, 3.1.E.8, 3.1.E.9, 3.1.E.10,
3.2.4, 3.2.5, 3.3, 3.4.1
- decision support system (DSS) - 1, 1.1, 1.2, 1.3, 1.5, 3.1
- environments - 1, 1.3, 1.4, 2.4
 - components - 1, 2.4
 - life cycle activity - 3.1.E.4
 - tool set - 3.1.E.4
- evaluation criteria - 1.2.1, 1.2.9, 1.3, 2.2, 3.1.E.3, 3.1.E.5, 3.1.E.7,
3.1.E.9, 3.3.1
- evaluation features - 1.2.1, 1.2.9, 1.3, 2.2, 3.1.E.3, 3.1.E.5, 3.1.E.6,
3.1.E.9, 3.3.1
- evaluation data - 1.2.8, 1.2.B, 1.3, 3.1.A, 3.1.E.9, 3.1.E.10, 3.3.1
- expert system - 1.3
- file formats - 3.1.A.1
- help - 3.1.E.1, 3.1.E.11, 3.1.E.12, 3.1.E.16
- hypertext - 1.3, 3.4.5
- input - 1.2.A, 1.2.B, 2.2
 - criteria selection - 2.2, 3.1.C, 3.1.D, 3.1.E.3, 3.1.E.5, 3.1.E.7, 3.1.E.8
 - conversion - 1.2.B, 3.1.A.1
 - evaluation information - 1.2.B, 1.2.D, 3.1.A
 - feature selection - 2.2, 3.1.C, 3.1.D, 3.1.E.3, 3.1.E.5, 3.1.E.6, 3.1.E.8
 - file - 3.1.A.1, 3.1.C.1, 3.1.C.3
 - interactive - 1.2.C, 2.5, 3.1.B.1, 3.1.B.2, 3.1.C.1, 3.1.C.2, 3.1.C.3, 3.2.1,
3.2.3
 - verification - 3.1.C.4
 - weights - 2.2, 3.1.E.6, 3.1.E.7

keywords - 3.1.A.1
knowledge base - 1.2, 1.2.B, 1.2.D, 2.2, 2.3, 3.1.B, 3.1.E.6, 3.1.E.7,
3.1.E.9, 3.2.5, 3.3, 3.4.1

levels of detail - 1.2, 2.2, 3.1.E.6, 3.1.E.7, 3.1.E.10

messages - 3.1.E.13, 3.1.E.14, 3.1.E.15
error - 3.1.C.4

output - 2.2, 3.1.D
file - 3.1.D.1
information - 1.2.2, 1.2.C, 3.1.D.2, 3.1.E.9, 3.1.E.10
printer - 3.1.D.2
recommendations - 1.2.2, 2.2, 3.1.E.9, 3.1.E.10, 3.2.3, 3.2.4
report - 1.2.5, 1.2.A, 3.1.D.2
statistical summaries - 1.2.2

reasoning process - 1.2.4, 3.1.E.12
response time - 3.2
retrieving - 1.2.D, 3.1.D.1

storing - 1.2.D, 3.1.D.1
subsystem interfaces - 2.4, 3.3.3
subsystems - 1.2, 2.4, 3.3.3
Decision Logic Subsystem - 1.2.C, 2.4, 3.3.2
Knowledge Acquisition Subsystem - 1.2.B, 3.1.A.3, 3.1.B.2, 3.4.1
Knowledge Base Subsystem - 1.2.B, 1.2.C, 1.2.D, 3.1.A.3, 3.1.A.4,
3.1.B.2, 3.1.B.3
User Interface Subsystem - 1.2.A, 1.2.B, 1.2.C, 3.1.A.3, 3.1.B.2

system manager - 1.2, 2.3, 3.4

user interface - 1.2.3, 2.5, 3.1.E.1, 3.1.E.2, 3.1.E.8, 3.1.E.13, 3.1.E.14,
3.1.E.15

windows - 1.2.A, 1.2.C, 1.3, 3.1.D.2, 3.1.E.1, 3.1.E.2, 3.1.E.16, 3.2.1,
3.2.2, 3.4

workstation - 2.1, 2.5, 3.4.6

Appendix D

Test Plan for ASSIST

1. System Description

This is a test plan for a decision support system (DSS) called the Ada Software Selection assISTant (ASSIST). ASSIST's purpose is to support decisions on the selection of Ada environments and their components, where the selection is based on the technical evaluation of the software.

1.1 Subsystems

The following subsystems are defined for ASSIST:

1) User Interface Subsystem

This subsystem provides the resources for developing the screen windows and reports and for capturing user input. This subsystem does not provide the logic for directing system processing.

2) Knowledge Acquisition Subsystem

This subsystem handles the acquisition of evaluation data for the database and the acquisition of criteria knowledge for the knowledge base. It accepts data or knowledge input via the User Interface Subsystem in any number of predefined formats, and it provides for the conversion of the input into the appropriate form for entry into the system database and knowledge base. It accesses the Knowledge Base Subsystem to store the input into the database or knowledge base.

3) Decision Logic Subsystem

This subsystem provides the main decision logic of the system. It provides appropriate windows for either collecting selection information from the user or providing information to the user. It accepts interactive input from the user via the User Interface Subsystem. It leads the user through the decision process, responding to user actions, and accessing the Knowledge Base Subsystem as necessary.

4) Knowledge Base Subsystem

This subsystem provides access to both the database and the knowledge base of the system. It is the only mechanism for storing and retrieving system information.

1.2 Targets

This system is targeted for a contemporary engineering workstation. The hardware available for the system is assumed to be a computer system which can provide adequate disk storage space for as much evaluation data as is available, acceptable execution speed, and a user interface which includes graphics capabilities as well as both keyboard and mouse input. The software will be implemented either in Ada or some higher level object oriented language. Adequate software tools which can support an interactive user interface using both text and graphics is assumed.

The initial specific target is the Macintosh II running HyperCard, an Ada development environment, and database software which is very efficient for large amounts of data. This is an immediate outgrowth of the system prototype, which is done entirely in HyperCard (using its HyperTalk language) on the Macintosh II.

2. Specific Test Objectives

The general testing approach is to first identify specific test objectives for each of the areas of correctness of system functions, performance, and general system attributes. From these test objectives, test cases are developed. These test cases are then combined into scenarios, and the scenarios are run as the system tests. Problems identified during system testing are fixed, and then regression testing follows.

In this chapter, the test objectives are identified and numbered. The organization is similar to that used for the system requirements.

2.1 Correctness of System Functions

Extensive testing is required to determine if a system is functioning correctly. For ASSIST, the functions to be tested are in the areas of database input, knowledge base input, user input, output, and system processing.

2.1.1 Database Input

Database input will be tested in the areas of input conversion, updates and deletions, access, consistency checking, and visibility to the decision maker.

2.1.1.1 Input Conversion

2.1.1.1.1 To determine if the database input is converted properly from at least 2 different formats.

2.1.1.1.2 To determine if the database input is converted properly from interactive input.

2.1.1.2 Updates and Deletions

2.1.1.2.1 To determine if database updates are accomplished properly.

2.1.1.2.2 To determine if database deletions are accomplished properly.

2.1.1.3 Access

2.1.1.3.1 To determine if the database is accessed properly.

2.1.1.4 Consistency Checking

2.1.1.4.1 To determine if the Knowledge Base Subsystem provides appropriate consistency checking for database input.

2.1.1.5 Visibility to Decision Maker

2.1.1.5.1 To verify that the system manager can get proper access to the database without the process being visible to the decision maker.

2.1.2 Knowledge Base Input

Knowledge base input will be tested in the areas of input, updates and deletions, access, consistency checking, and visibility to the decision maker.

2.1.2.1 Input

2.1.2.1.1 To determine if the interactive knowledge base input is properly handled.

2.1.2.2 Updates and Deletions

2.1.2.2.1 To determine if knowledge base updates are accomplished properly.

2.1.2.2.2 To determine if knowledge base deletions are accomplished properly.

2.1.2.3 Access

2.1.2.3.1 To determine if the knowledge base is accessed properly.

2.1.2.4 Consistency Checking

2.1.2.4.1 To determine if the Knowledge Base Subsystem provides appropriate consistency checking for knowledge base input.

2.1.2.5 Visibility to Decision Maker

2.1.2.5.1 To verify that the system manager can get proper access to the knowledge base without the process being visible to the decision maker.

2.1.3 User Input

User input will be tested in the areas of method and the order of selections.

2.1.3.1 Method

2.1.3.1.1 To determine if valid user inputs are processed correctly.

2.1.3.1.2 To determine if the user can save all selections made in the software selection process.

2.1.3.1.3 To determine if the saved selections can be retrieved and then used as if they had been selected interactively.

2.1.3.2 Order of Selections

2.1.3.2.1 To determine if selections can be made out of the normal order.

2.1.3.2.2 To determine if changes can be made to the user selections.

2.1.4 Output

Output will be tested in the areas of printer output, summary information displayed in a window on the screen, and selections saved to disk.

2.1.4.1 Printer

2.1.4.1.1 To determine if the entire window on the screen can be printed.

2.1.4.1.2 To determine if a field in the window on the screen can be printed.

2.1.4.2. Displayed Summary Information

2.1.4.2.1 To determine if the system will correctly not produce summary information until all selections have been properly made.

2.1.4.2.2 To determine if the system will correctly produce summary recommendation information at the highest level of output detail.

2.1.4.2.3. To determine if the system will correctly produce analysis information at the second level of detail.

2.1.4.2.4 To determine if the system will correctly produce lower level information at successively lower levels of detail.

2.1.4.2.5 To determine if the system will correctly produce evaluation data at the lowest level of detail.

2.1.4.3 Selections Saved to Disk

2.1.4.3.1 To determine if the saved selections are saved to disk for permanent storage.

2.1.5 System Processing

System processing will be tested by testing the help system, the reasoning process option, and each of the activities which occurs during the normal process of making selections and obtaining the resulting summary display.

2.1.5.1 Help System

2.1.5.1.1 To determine if the system provides adequate and appropriate explanations at the beginning of the program execution.

2.1.5.1.2 To determine if adequate and appropriate help is always available during the selection process.

2.1.5.1.3 To determine if a graphical browser is always available during processing to show the current location and to permit changing location in the selection process.

2.1.5.2 Reasoning Process Option

2.1.5.2.1 To determine if the reasoning process option properly reviews the selections which have been made and, if appropriate, provides a summary of the results.

2.1.5.3 Normal System Processing

Normal system processing will be tested by testing the areas of software type selection, application area selection, feature selections, criteria selections, making choices, and consistency.

2.1.5.3.1 Software Type Selection

2.1.5.3.1.1 To determine if selecting the software type is properly handled as a part of the selection process.

2.1.5.3.2 Application Area Selection

2.1.5.3.2.1 To determine if selecting the application area is properly handled as a part of the selection process.

2.1.5.3.3 Feature Selections

2.1.5.3.3.1 To determine if the suggested feature selections are appropriate for the selected application area.

2.1.5.3.3.2 To determine if multiple levels of features selections are available.

2.1.5.3.3.3 To determine if the finest level of detail available for feature selections is consistent with the knowledge base.

2.1.5.3.3.4 To determine if features may be added to the suggested feature selections.

2.1.5.3.3.5 To determine if default weights are given to feature selections and if the user has the option to change them.

2.1.5.3.4 Criteria Selections

2.1.5.3.4.1 To determine if the suggested criteria selections are appropriate for the selected application area.

2.1.5.3.4.2 To determine if multiple levels of criteria selections are available.

2.1.5.3.4.3 To determine if the finest level of detail available for criteria selections is consistent with the knowledge base.

2.1.5.3.4.4 To determine if criteria may be added to the suggested criteria selections.

2.1.5.3.4.5 To determine if default weights are given to criteria selections and if the user has the option to change them.

2.1.5.3.5 Making Choices

2.1.5.3.5.1 To determine if the user is usually presented with choices which require only either a mouse click or a single keystroke to select rather than being required to type responses.

2.1.5.3.5.2 To determine if the user is not restricted by the simplified selection process.

2.1.5.3.5.3 To determine if the choices given to the user are valid and consistent with the knowledge base.

2.1.5.3.6 Consistency

2.1.5.3.6.1 To determine if the output summary is consistent with the data in the database and the knowledge in the knowledge base.

2.2 Performance

Performance is an important part of any system. For this system, the performance test objectives fall into the categories of response times and size constraints.

2.2.1 Response Times

2.2.1.1 To determine if the average response time is adequate when the user provides an input which results in a system action.

2.2.1.2 To determine if the response time is adequate when displaying a window.

2.2.1.3 To determine if the response time is adequate when the system is doing summary processing.

2.2.2 Size Constraints

2.2.2.1 To determine if the system handles constraints on the size of the stored database and knowledge base adequately.

2.3 General System Attributes

For ASSIST, general system attributes fall into the categories of usability, reliability, and subsystem structure.

2.3.1 Usability

2.3.1.1 To determine if ASSIST will start up properly from a "foreign" system (not one used for its development).

2.3.1.2 To determine if each instruction is clear to the casual user.

2.3.1.3 To determine if the interactive user procedures for the software selection process are simple, direct, and correct.

2.3.1.4 To determine if the user receives appropriate feedback after each action taken.

2.3.1.5 To determine if the user can quit from the program easily, but with appropriate reminders, to preclude inadvertent loss of information.

2.3.1.6 To determine if the system meets the minimum standards for system output display.

2.3.2 Reliability

2.3.2.1 To determine if the system correctly recognizes error conditions.

2.3.2.2 To determine if the system produces appropriate messages for error conditions encountered.

2.3.2.3 To determine if the system continues execution after an error condition has occurred, if appropriate.

2.3.2.4 To evaluate system integrity if execution continues after an error condition.

2.3.3 Subsystem Structure

2.3.3.1 To determine if the system is structured using the correct subsystems.

2.3.3.2 To determine if the subsystem interfaces are the only means of communication among the subsystems.

2.4 Scenarios

Each of the test scenarios will test a combination of test cases generated from the above test objectives. Hence, the objectives for the scenarios are more general statements representing combinations of the above test objectives.

2.4.1 To determine if the system manager capabilities for loading the database and knowledge base work properly.

2.4.2 To determine if the user selection and summary display functions work correctly when making no changes to the "normal" process of making selections.

2.4.3 To determine if the help system and the general and auxiliary functions work correctly, and if performance is adequate.

2.4.4 To determine if the user selection and summary display functions work correctly when making changes, including changing the order of selections.

2.4.5 To inspect the code for adherence to structuring constraints.

2.5 Regression Testing

Regression testing will be performed as necessary after problems have been encountered with the tests of the scenarios. An attempt will be made to test as much as possible, and at least one full scenario, during each testing session. The lead tester will decide the extent of regression testing which is necessary after any particular scenario has been tested and software changes have subsequently been made in the areas tested.

3. References

- [1] P. K. Lawlis, *Requirements for Ada Software Selection assISant (ASSIST)*, produced as a part of PhD dissertation work, Arizona State University, 9 March 1989.
- [2] P. K. Lawlis, *Supporting Selection Decisions Based on the Technical Evaluation of Ada Environments and Their Components*, PhD Dissertation Proposal, Arizona State University, 28 October 1988.

Appendix E

Test Descriptions

Test	
Test Number	<u>2.1.1.1.1</u>
Test Objective	To determine if the database input is converted properly from at least 2 different formats.
Test Description	Data should be loaded from files using 2 different formats, each of which is different from the format used by the system database. The database should then be inspected to ensure that the data is properly converted. The system should also be run once using the new database to ensure that no unusual problems occur when accessing the database.
Requirements	<u>3.1.A.1, 3.1.A.3</u>
Design Reference	<u>3</u>
	

Test	
Test Number	<u>2.1.1.1.2</u>
Test Objective	To determine if the database input is converted properly from interactive input.
Test Description	Data should be loaded from interactive input. The database should then be inspected to ensure that the data is properly converted. The system should also be run once using the new database to ensure that no unusual problems occur when accessing the database.
Requirements	<u>3.1.A.1, 3.1.A.3</u>
Design Reference	<u>3</u>
	

Tests	
Test Number	<u>2.1.1.2.1</u>
Test Objective	To determine if database updates are accomplished properly.
Test Description	Some database information should be changed interactively. The database should then be inspected for proper changes, and the system should be run once using the new database to ensure that no unusual problems occur when accessing the database.
Requirements	<u>3.1.A.2, 3.1.A.3</u>
Design Reference	<u>2, 2.2, 3</u>
	

Tests	
Test Number	<u>2.1.1.2.2</u>
Test Objective	To determine if database deletions are accomplished properly.
Test Description	Some database information should be deleted interactively. The database should then be inspected for proper changes, and the system should be run once using the new database to ensure that no unusual problems occur when accessing the database.
Requirements	<u>3.1.A.2, 3.1.A.3</u>
Design Reference	<u>2, 2.2, 3</u>
	

Test	
Test Number	<u>2.1.1.3.1</u>
Test Objective	To determine if the database is accessed properly.
Test Description	Code must be inspected to ensure that no database access is being made except through the interface defined by the Knowledge Base Subsystem.
Requirements	<u>3.1.A.3</u>
Design Reference	<u>2. 2.2</u>

Test	
Test Number	<u>2.1.1.4.1</u>
Test Objective	To determine if the Knowledge Base Subsystem provides appropriate consistency checking for database input.
Test Description	The attempt should be made to enter inconsistent data into the system database. This should be rejected by the system, and it should ask for user response to determine which of the conflicting input to accept. If the new input is accepted, the old should be discarded. This should be verified by an inspection of the database, and the system should be run once using the new database to ensure that no unusual problems occur when accessing the database.
Requirements	<u>3.1.A.4</u>
Design Reference	<u>2. 2.2</u>

Tests	
Test Number	<u>2.1.1.5.1</u>
Test Objective	To verify that the system manager can get proper access to the database without the process being visible to the decision maker.
Test Description	Data should be entered into the database both interactively and from a file. This process should use the Knowledge Acquisition Subsystem, but it should be via a mechanism not visible to the normal user.
Requirements	<u>3.4.1</u>
Design Reference	<u>3</u>

Tests	
Test Number	<u>2.1.2.1.1</u>
Test Objective	To determine if the knowledge in the knowledge base is properly handled.
Test Description	Knowledge should be loaded into the knowledge base from interactive input. The knowledge base should then be inspected to ensure that the data is properly entered. The system should also be run once using the new knowledge base to ensure that no unusual problems occur when accessing the knowledge base.
Requirements	<u>3.1.B.1, 3.1.B.2</u>
Design Reference	<u>2, 2.1, 2.1.1, 2.1.2, 3</u>

Tests	
Test Number	<u>2.1.2.2.1</u>
Test Objective	To determine if knowledge base updates are accomplished properly.
Test Description	Some knowledge base information should be changed interactively. The knowledge base should then be inspected for proper changes, and the system should be run once using the new knowledge base to ensure that no unusual problems occur when accessing the knowledge base.
Requirements	<u>3.1.B.1, 3.1.B.2</u>
Design Reference	<u>2, 2.1, 3</u>

Tests	
Test Number	<u>2.1.2.2.2</u>
Test Objective	To determine if knowledge base deletions are accomplished properly.
Test Description	Some knowledge base information should be deleted interactively. The knowledge base should then be inspected for proper changes, and the system should be run once using the new knowledge base to ensure that no unusual problems occur when accessing the knowledge base.
Requirements	<u>3.1.B.1, 3.1.B.2</u>
Design Reference	<u>2, 2.1, 3</u>

Tests	
Test Number	<u>2.1.2.3.1</u>
Test Objective	To determine if the knowledge base is accessed properly.
Test Description	Code must be inspected to ensure that no knowledge base access is being made except through the interface defined by the Knowledge Base Subsystem.
Requirements	<u>3.1.B.2</u>
Design Reference	<u>2. 2.1</u>
	

Tests	
Test Number	<u>2.1.2.4.1</u>
Test Objective	To determine if the Knowledge Base Subsystem provides appropriate consistency checking for knowledge base input.
Test Description	The attempt should be made to enter inconsistent data into the system knowledge base. This should be rejected by the system, and it should ask for user response to determine which of the conflicting input to accept. If the new input is accepted, the old should be discarded. This should be verified by an inspection of the knowledge base, and the system should be run once using the new knowledge base to ensure that no unusual problems occur when accessing the knowledge base.
Requirements	<u>3.1.B.3</u>
Design Reference	<u>2. 2.1</u>
	

Test	
Test Number	<u>2.1.2.5.1</u>
Test Objective	To verify that the system manager can get proper access to the knowledge base without the process being visible to the decision maker.
Test Description	Knowledge should be entered into the knowledge base interactively. This process should use the Knowledge Acquisition Subsystem, but it should be via a mechanism not visible to the normal user.
Requirements	<u>3.4.1</u>
Design Reference	<u>3</u>
	

Test	
Test Number	<u>2.1.3.1.1</u>
Test Objective	To determine if valid user inputs are processed correctly.
Test Description	In 3 separate runs through the system processing, 3 separate sets of valid selections should be entered. These should be accepted by the system, and the resulting summaries should be checked against the data in the database and the knowledge in the knowledge base to assure that the results are consistent with this information. The first set of selections should be chosen to represent "easy" selections, using defaults made available by the system. The second and third sets should require making changes to the defaults. The test is passed only if there are no discrepancies found.
Requirements	<u>3.1.C.4</u>
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6</u>
	

Test	
Test Number	2.1.3.1.2
Test Objective	To determine if the user can save all selections made in the software selection process.
Test Description	After a selection process has been completed interactively, the selections should be saved using the system option, and the saved options should be noted. The selection process should then be restarted, and the saved selections should be retrieved. The system should be able to process a summary and display the exact same results as before, without any selection information being repeated interactively. A check of the selections should also indicate the exact same ones which had been saved.
Requirements	3.1.D.1
Design Reference	1.2, 1.2.6, 1.4, 1.4.2
	

Test	
Test Number	2.1.3.1.3
Test Objective	To determine if the saved selections can be retrieved and then used as if they had been selected interactively.
Test Description	When a saved selection file is retrieved, the user should attempt to make interactive changes to the information before processing a summary. These changes should then be reflected in the summary results obtained.
Requirements	3.1.C.1, 3.1.C.2, 3.1.C.3
Design Reference	1.2, 1.2.4, 1.2.5, 1.2.6
	

Test	
Test Number	<u>2.1.3.2.1</u>
Test Objective	To determine if selections can be made out of the normal order.
Test Description	While making a set of interactive selections, the user should attempt to enter selections out of order. The system should permit the out-of-order selection if no order dependence problem exists. Otherwise, if an order dependence has not been satisfied, it should gracefully indicate that something else must be done first. The test is passed if all selections can be made without the system hanging, crashing, or otherwise failing, and the summary is provided.
Requirements	<u>3.1.C.2</u>
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6, 1.3</u>
	

Test	
Test Number	<u>2.1.3.2.2</u>
Test Objective	To determine if changes can be made to the user selections.
Test Description	After making a set of interactive selections and getting a summary, the user should go back and change some of those selections and then get another summary. The changes should then be reflected in the summary results obtained.
Requirements	<u>3.1.C.2</u>
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6, 1.3</u>
	

Tests	
Test Number	<u>2.1.4.1.1</u>
Test Objective	To determine if the entire window on the screen can be printed.
Test Description	Use the print option to print the window. The test is passed if the window prints exactly as it looks on the screen.
Requirements	<u>3.1.D.2</u>
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.3</u>
	

Tests	
Test Number	<u>2.1.4.1.2</u>
Test Objective	To determine if a field in the window on the screen can be printed.
Test Description	Use the option to print the contents of a field in the active window. The test is passed if the printed results are identical to the information in the field.
Requirements	<u>3.1.D.2</u>
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.3</u>
	

Test	
Test Number	2.1.4.2.1
Test Objective	To determine if the system will correctly not produce summary information until all selections have been properly made.
Test Description	Attempt to generate summary results with no selections made. If this is rejected, perform one selection step at a time, not in the normal order, followed by another attempt to generate summary results, until all selections are made. The test is passed if summary results are not produced until all selections are made.
Requirements	3.1.D.3, 3.1.E.3, 3.1.E.9
Design Reference	1.2, 1.2.6

Test	
Test Number	2.1.4.2.2
Test Objective	To determine if the system will correctly produce summary recommendation information at the highest level of output detail.
Test Description	For each of 3 sets of selections, check that the first level of summary results gives recommendations without further details. This test is passed if this is found. This test is concerned with the correct output form as opposed to correct content.
Requirements	3.1.D.3
Design Reference	1.2, 1.2.6, 1.5, 1.5.4, 2.1.4, 2.1.4.1, 2.1.4.2, 2.1.4.3, 2.1.4.4, 2.1.4.5

Analysis	
Test Number	<u>2.1.4.2.3</u>
Test Objective	To determine if the system will correctly produce analysis information at the second level of detail.
Test Description	For each of 3 sets of selections, check the second level of summary results for analysis information beyond the recommendations provided in the first level. This analysis should include an explanation of the formulas used for producing the recommendations made, and it should provide the user with the ability to change parameters in these formulas. This test is passed if this is found.
Requirements	<u>3.1.D.3</u>
Design Reference	<u>1.2, 1.2.6, 1.5, 1.5.1, 1.5.2, 1.5.4, 2.1.4, 2.1.4.1, 2.1.4.2, 2.1.4.3, 2.1.4.4, 2.1.4.5</u>

Analysis	
Test Number	<u>2.1.4.2.4</u>
Test Objective	To determine if the system will correctly produce lower level information at successively lower levels of detail.
Test Description	For each of 3 sets of selections, check beyond the second level of summary results for successively detailed results explaining the process which led to the recommendations made. This test is passed if existing successive levels have more detailed results. At least one level beyond the second level must exist.
Requirements	<u>3.1.D.3</u>
Design Reference	<u>1.2, 1.2.6, 1.5, 1.5.1, 1.5.2, 1.5.4, 2.1.4, 2.1.4.1, 2.1.4.2, 2.1.4.3, 2.1.4.4, 2.1.4.5</u>

Tests	
Test Number	<u>2.1.4.2.5</u>
Test Objective	To determine if the system will correctly produce evaluation data at the lowest level of detail.
Test Description	For each of 3 sets of selections, check the lowest level of summary results against the information in the database for the evaluations which were considered. This test is passed if the results at this level match the database information.
Requirements	<u>3.1.E.10</u>
Design Reference	<u>1.2, 1.2.6, 1.5, 1.5.4</u>
	

Tests	
Test Number	<u>2.1.4.3.1</u>
Test Objective	To determine if the saved selections are saved to disk for permanent storage.
Test Description	After a selection process has been completed interactively, the selections should be saved using the system option, and the saved options should be noted. The system should then be quit and restarted, and the saved selections should be retrieved. The system should be able to process a summary and display the exact same results as before, without any selection information being repeated interactively. A check of the selections should also indicate the exact same ones which had been saved.
Requirements	<u>3.1.D.1</u>
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.2</u>
	

Tests	
Test Number	<u>2.1.5.1.1</u>
Test Objective	To determine if the system provides adequate and appropriate explanations at the beginning of the program execution.
Test Description	Start the system, and check that explanations are given at the very beginning about conventions used and how the system works. If these are not automatically given, it should be clear to the user that a single action will cause this information to be available.
Requirements	<u>3.1.E.1</u>
Design Reference	<u>1.2, 1.2.6</u>

Tests	
Test Number	<u>2.1.5.1.2</u>
Test Objective	To determine if adequate and appropriate help is always available during the selection process.
Test Description	At each step of the selection process, attempt to access help. Specific help for the current window should be available with a single action on the part of the user.
Requirements	<u>3.1.E.1, 3.1.E.16</u>
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.1</u>

Tests	
Test Number	<u>2.1.5.1.3</u>
Test Objective	To determine if a graphical browser is always available during processing to show the current location and to permit changing location in the selection process.
Test Description	At each step of the selection process, attempt to access the graphical browser. It should always be available with a single action on the part of the user, and it should always give an indication of what the current window is and permit changing to a different window in the selection process. Attempt to change location at least twice.
Requirements	<u>3.1.E.11</u>
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.5</u>
	

Tests	
Test Number	<u>2.1.5.2.1</u>
Test Objective	To determine if the reasoning process option properly reviews the selections which have been made.
Test Description	While executing the selection process of the program, select the option for reviewing the reasoning process at least 3 times at different steps. Each time the information given should match the selections actually made to this point. After the summary has been provided, select the review again. It should now indicate the full set of selections, and it should refer to the summary output for as much detail as is desired.
Requirements	<u>3.1.E.12</u>
Design Reference	<u>1.2, 1.2.6, 1.3, 1.4, 1.4.4</u>
	

Test	
Test Number	<u>2.1.5.3.1.1</u>
Test Objective	To determine if selecting the software type is properly handled as a part of the selection process.
Test Description	The system should require the selection of the software type. The user should have choices available to choose from. An attempt should be made to select a software type for which the database contains no information. The system should gracefully indicate the problem and require a different selection. A software type for which data exists in the database should be accepted.
Requirements	<u>3.1.E.4</u>
Design Reference	<u>1.2, 1.2.2, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.2.1</u>
Test Objective	To determine if selecting the application area is properly handled as a part of the selection process.
Test Description	The system should require the selection of the application area. The user should have choices available to choose from.
Requirements	<u>3.1.E.5</u>
Design Reference	<u>1.2, 1.2.3, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.3.1</u>
Test Objective	To determine if the suggested feature selections are appropriate for the selected application area.
Test Description	The system should require the selection of the software features. Default features should be provided, and these should be consistent with the features indicated for the given application area in the knowledge base.
Requirements	<u>3.1.E.5</u>
Design Reference	<u>1.2, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.3.2</u>
Test Objective	To determine if multiple levels of feature selections are available.
Test Description	After selections are completed for the first level of features, the user should have an option to describe these selections in more detail.
Requirements	<u>3.1.E.6</u>
Design Reference	<u>1.2, 1.2.4, 1.2.6</u>
	

Tests	
Test Number	<u>2.1.5.3.3.3</u>
Test Objective	To determine if the finest level of detail available for feature selections is consistent with the knowledge base.
Test Description	Selections should be made at as many levels of detail as provided by the system. Once no more detail is possible, the resulting detailed feature selections should be compared with the knowledge in the knowledge base for the selected features. The test is passed if this knowledge is consistent.
Requirements	<u>3.1.E.6</u>
Design Reference	<u>1.2, 1.2.4, 1.2.6</u>
	

Tests	
Test Number	<u>2.1.5.3.3.4</u>
Test Objective	To determine if features may be added to the suggested feature selections.
Test Description	An attempt should be made to add a feature to the list of suggested feature selections. The system should provide a list of features at the current level of detail which are recognized by the system, and from which the user may choose. This list should be compared against the features in the knowledge base for consistency. A choice should be made and accepted by the system. The process should be repeated at least once more.
Requirements	<u>3.1.E.6, 3.1.E.8</u>
Design Reference	<u>1.2, 1.2.4, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.3.5</u>
Test Objective	To determine if default weights are given to feature selections and if the user has the option to change them.
Test Description	Default weights should be provided for each of the features selected. An attempt should be made to change at least two of them. The test is passed if both can be changed.
Requirements	<u>3.1.E.6</u>
Design Reference	<u>1.2, 1.2.4, 1.2.6</u>

Test	
Test Number	<u>2.1.5.3.4.1</u>
Test Objective	To determine if the suggested criteria selections are appropriate for the selected application area.
Test Description	The system should require the selection of the software criteria. Default criteria should be provided, and these should be consistent with the criteria indicated for the given application area in the knowledge base.
Requirements	<u>3.1.E.5</u>
Design Reference	<u>1.2, 1.2.6</u>

Tests	
Test Number	<u>2.1.5.3.4.2</u>
Test Objective	To determine if multiple levels of criteria selections are available.
Test Description	After selections are completed for the first level of criteria, the user should have an option to describe these selections in more detail.
Requirements	<u>3.1.E.7</u>
Design Reference	<u>1.2, 1.2.5, 1.2.6</u>
	

Tests	
Test Number	<u>2.1.5.3.4.3</u>
Test Objective	To determine if the finest level of detail available for criteria selections is consistent with the knowledge base.
Test Description	Selections should be made at as many levels of detail as provided by the system. Once no more detail is possible, the resulting detailed criteria selections should be compared with the knowledge in the knowledge base for the selected criteria. The test is passed if this knowledge is consistent.
Requirements	<u>3.1.E.7</u>
Design Reference	<u>1.2, 1.2.5, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.4.4</u>
Test Objective	To determine if criteria may be added to the suggested criteria selections.
Test Description	An attempt should be made to add a criterion to the list of suggested criteria selections. The system should provide a list of criteria at the current level of detail which are recognized by the system, and from which the user may choose. This list should be compared against the criteria in the knowledge base for consistency. A choice should be made and accepted by the system. The process should be repeated at least once more.
Requirements	<u>3.1.E.7, 3.1.E.8</u>
Design Reference	<u>1.2, 1.2.5, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.4.5</u>
Test Objective	To determine if default weights are given to criteria selections and if the user has the option to change them.
Test Description	Default weights should be provided for each of the criteria selected. An attempt should be made to change at least two of them. The test is passed if both can be changed.
Requirements	<u>3.1.E.7</u>
Design Reference	<u>1.2, 1.2.5, 1.2.6</u>
	

Tests	
Test Number	<u>2.1.5.3.5.1</u>
Test Objective	To determine if the user is usually presented with choices which require only either a mouse click or a single keystroke to select rather than being required to type responses.
Test Description	The user should go through the entire selection process, and for each selection, choices should be available which require no typing except possibly one keystroke.
Requirements	<u>3.1.E.8</u>
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6</u>
	

Tests	
Test Number	<u>2.1.5.3.5.2</u>
Test Objective	To determine if the user is not restricted by the simplified selection process.
Test Description	For each selection made in the process of executing the program, either all possible selections should be available, or else the user should have the option of changing to or adding any selections which are not available as defaults.
Requirements	<u>3.1.E.8</u>
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6</u>
	

Test	
Test Number	<u>2.1.5.3.5.3</u>
Test Objective	To determine if the choices given to the user are valid and consistent with the knowledge base.
Test Description	For each selection choice made available in the selection process, choices should be found in the knowledge base and their selection for that purpose should be consistent with the knowledge given in the knowledge base. This must be verified by inspecting the contents of the knowledge base.
Requirements	<u>3.1.E.8</u>
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6</u>

Test	
Test Number	<u>2.1.5.3.6.1</u>
Test Objective	To determine if the output summary is consistent with the data in the database and the knowledge in the knowledge base.
Test Description	For each output summary resulting from 3 different sets of selections, recommendations and information provided should be consistent with the data in the database and the knowledge given in the knowledge base. One of these sets of selections should be entered by interactively running through the selection process in the "normal" order. Another set should be entered by making the selections out of the "normal" order. The third set should be retrieved after having been saved from a previous session. The summary results must be verified by inspecting the contents of the database and the knowledge base.
Requirements	<u>3.1.E.9, 3.3.1, 3.3.2</u>
Design Reference	<u>1.2, 1.2.6, 1.5, 1.5.1, 1.5.2, 1.5.4, 2.1, 2.1.3, 2.1.3.1, 2.1.3.2, 2.1.4, 2.1.4.1, 2.1.4.2, 2.1.4.3, 2.1.4.4, 2.1.4.5</u>

Tests	
Test Number	<u>2.2.1.1</u>
Test Objective	To determine if the average response time is adequate when the user provides an input which results in a system action.
Test Description	For a moderate sized database and at least 3 different sets of selections, a stopwatch should be started at the time each action in the selection process is started and stopped when it has completed. The test is passed if the average time for each action, except for the final display of summary results, is less than 5 seconds.
Requirements	<u>3.2.1, 3.2.3</u>
Design Reference	<u>1.2, 1.2.6</u>

Tests	
Test Number	<u>2.2.1.2</u>
Test Objective	To determine if the response time is adequate when displaying a window.
Test Description	A stop watch should be started at the time the first pixel of a new window is displayed and stopped when the last pixel is displayed. If the elapsed time is less than 2 seconds, the test is passed.
Requirements	<u>3.2.2</u>
Design Reference	<u>1.5.3</u>

Tests	
Test Number	<u>2.2.1.3</u>
Test Objective	To determine if the response time is adequate when the system is doing summary processing.
Test Description	For a moderate sized database and at least 3 different sets of selections, the display should be monitored as the system processes the summary results. The window to contain the results should be displayed within 5 seconds. A new message, indicating some progress, should be displayed at least every 30 seconds until the processing is finished. The database should be examined to determine the number of evaluations which should have been considered for each set of selections. The total processing time for each selection should not exceed a number of seconds equal to the number of evaluations processed for the selection.
Requirements	<u>3.2.1, 3.2.4</u>
Design Reference	<u>1.2, 1.2.6, 1.5.4</u>

Tests	
Test Number	<u>2.2.2.1</u>
Test Objective	To determine if the system handles constraints on the size of the stored database and knowledge base adequately.
Test Description	An attempt should be made to load into the system database and knowledge base an amount of data exceeding the system storage capacity. The system storage capacity should be changed, and the same attempt tried again. The system should gracefully reject the amount which cannot be stored, and this should be a different amount in each case. In addition, the code should be inspected to ensure that a limit to the size of the database and knowledge base is not hard-coded.
Requirements	<u>3.2.5</u>
Design Reference	<u>2</u>

Tests	
Test Number	2.3.1.1
Test Objective	To determine if ASSIST will start up properly from a "foreign" system (not one used for its development).
Test Description	Transfer the ASSIST file to the system to be used for testing, ensuring that no other development files are on this system. Run ASSIST through the process of making selections and getting summary results. Select help, the reasoning process, and the compass. Save the selections, restart the program, and run through the process again, but this time retrieve the selections from a file. The test is passed if no problems are encountered.
Requirements	3.4.6
Design Reference	4
	

Tests	
Test Number	2.3.1.2
Test Objective	To determine if each instruction is clear to the casual user.
Test Description	Have someone who is not a programmer and who knows nothing about the construction of the program read the instructions on the windows seen by the casual user, having the user manual also available. The instructions should be understood. If not, they should be changed to become clear, basing changes on comments provided by the reviewer.
Requirements	3.1.E.1, 3.1.E.3
Design Reference	1.2, 1.2.1, 1.2.6
	

Tests	
Test Number	<u>2.3.1.3</u>
Test Objective	To determine if the interactive user procedures for the software selection process are simple, direct, and correct.
Test Description	Have someone who is not a programmer and who knows nothing about the construction of the program read the instructions on the windows seen by the casual user, having the user manual also available, and follow the instructions. The test is passed if the user always has available an option to either choose from given selections, select a given action, or input text, if correct summary results are obtained, and if the process created no particular difficulties for the user.
Requirements	<u>3.1.C.1, 3.1.E.2, 3.1.E.3, 3.4.5</u>
Design Reference	<u>1.1.1, 1.1.2, 1.2, 1.2.1, 1.2.4, 1.2.5, 1.2.6, 1.4, 1.4.6</u>
	

Tests	
Test Number	<u>2.3.1.4</u>
Test Objective	To determine if the user receives appropriate feedback after each action taken.
Test Description	While running through the selection process, check that for each delay an indication is given to the user that the system is working. If the delay is more than 5 seconds, a message should be provided.
Requirements	<u>3.1.E.13, 3.4.2, 3.4.3</u>
Design Reference	<u>1.2, 1.2.6, 1.5.4, 4</u>
	

Test	
Test Number	<u>2.3.1.5</u>
Test Objective	To determine if the user can quit from the program easily, but with appropriate reminders, to preclude inadvertant loss of information.
Test Description	At each step along the selection process, select the option to quit. If selections have been made and not saved, the option should be given to the user to save the selections before quitting.
Requirements	<u>3.1.E.14</u>
Design Reference	<u>1.2, 1.2.6, 1.4</u>
	

Test	
Test Number	<u>2.3.1.6</u>
Test Objective	To determine if the system meets the minimum standards for system output display.
Test Description	While the selection process is being executed, note whether at least one window is always displayed on the screen, the active window is always clearly marked, and both graphics and text are used in the displays. The test is passed if all of these conditions are true.
Requirements	<u>3.1.E.2, 3.4.2, 3.4.3, 3.4.4</u>
Design Reference	<u>1.5, 1.5.3, 4</u>
	

Tests	
Test Number	<u>2.3.2.1</u>
Test Objective	To determine if the system correctly recognizes error conditions.
Test Description	Run through the selection process several times, mixing up the order of the selections for type of software, application area, feature choices, feature weights, criteria choices, and criteria weights. Attempt to enter weights outside the specified range. The test is passed if only valid input was accepted.
Requirements	<u>3.1.C.4</u>
Design Reference	<u>1.2, 1.2.6</u>

Tests	
Test Number	<u>2.3.2.2</u>
Test Objective	To determine if the system produces appropriate messages for error conditions encountered.
Test Description	Run through the selection process several times, mixing up the order of the selections for type of software, application area, feature choices, feature weights, criteria choices, and criteria weights. Attempt to enter weights outside the specified range. The test is passed if meaningful error messages were given in all cases where processing could not continue in the normal order.
Requirements	<u>3.1.C.4, 3.1.E.15</u>
Design Reference	<u>1.2, 1.2.6</u>

Tests	
Test Number	<u>2.3.2.3</u>
Test Objective	To determine if the system continues execution after an error condition has occurred, if appropriate.
Test Description	When a condition is encountered which does not permit processing to proceed normally, a message should be given indicating the problem, and then the system should continue running unless the error is catastrophic. The test is passed if all selections made to the point where normal processing cannot continue are still in effect, and the system is in a state where processing may continue, selections may be completed, and summary results may be given.
Requirements	<u>3.1.C.4, 3.1.E.15</u>
Design Reference	<u>1.2, 1.2.6</u>
	

Tests	
Test Number	<u>2.3.2.4</u>
Test Objective	To evaluate system integrity if execution continues after an error condition.
Test Description	After an error condition has been encountered and processing has continued, changes should be made to some, but not all, of the selections made prior to the error condition. Then the selection process should be completed. The test is passed if the summary results are consistent with the selections made (as altered) and consistent with the database and knowledge base.
Requirements	<u>3.1.E.15</u>
Design Reference	<u>1.2, 1.2.6</u>
	

Tests	
Test Number	<u>2.3.3.1</u>
Test Objective	To determine if the system is structured using the correct subsystems.
Test Description	The code will be inspected to determine if the subsystems defined are the Knowledge Acquisition Subsystem, the User Interface Subsystem, the Decision Logic Subsystem, and the Knowledge Base Subsystem.
Requirements	<u>3.1.A.3, 3.1.B.2, 3.3.2, 3.3.3</u>
Design Reference	<u>0, 1, 2, 3, 4</u>
  	

Tests	
Test Number	<u>2.3.3.2</u>
Test Objective	To determine if the subsystem interfaces are the only means of communication among the subsystems.
Test Description	Code inspection should determine that clear interfaces are defined for using the resources from each subsystem. Further code inspection of each subsystem should determine that one does not use the resources of another without using the defined interfaces.
Requirements	<u>3.3.3</u>
Design Reference	<u>0, 1, 2, 3, 4</u>
  	

Tests	
Scenario Number	2.4.1
Scenario Objective	To determine if the system manager capabilities for loading the database and knowledge base work properly.
Scenario Description	Tests 2.3.1.1 and 2.1.1.5.1 can be checked when the system is first activated, then the remaining 2.1.1 database tests can be run in order. Next test 2.1.2.5.1 can be checked when activating the system for knowledge base access, and the remaining 2.1.2 knowledge base tests can be run in order. Finally, test 2.2.2.1 can be run. This scenario will be very time consuming because, with the exception of 2.3.1.1, 2.1.1.5.1, and 2.1.2.5.1, each must be tested with a separate run of the program. Database and knowledge base examination are also required.
 Tests	2.1.1.1.1, 2.1.1.1.2, 2.1.1.2.1, 2.1.1.2.2, 2.1.1.4.1, 2.1.1.5.1, 2.1.2.1.1, 2.1.2.2.1, 2.1.2.2.2, 2.1.2.4.1, 2.1.2.5.1, 2.2.2.1, 2.3.1.1
 	

Tests	
Scenario Number	2.4.2
Scenario Objective	To determine if the user selection and summary display functions work correctly when making no changes to the "normal" process of making selections.
Scenario Description	This scenario requires running the system completely through 3 times, with 3 entirely different sets of selections, and with most tests checked each run. On the first run, no lower levels of detail should be chosen for either features or criteria and no default weights should be changed. For the second run, weights should be changed, but still no lower levels of detail should be chosen for either features or criteria. For the third run, lower levels of detail should be chosen (tests 2.1.5.3.3.2 and 2.1.5.3.4.2 are only tested in this run). Each run requires examination of the database and knowledge base, and response timing should be done for the summary display during each run.
 Tests	2.1.3.1.1, 2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.1.1, 2.1.5.3.2.1, 2.1.5.3.3.1, 2.1.5.3.3.2, 2.1.5.3.3.3, 2.1.5.3.3.5, 2.1.5.3.4.1, 2.1.5.3.4.2, 2.1.5.3.4.3, 2.1.5.3.4.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.1
 	

Tests	
Scenario Number	<u>2.4.3</u>
Scenario Objective	To determine if the help system and the general and auxiliary functions work correctly, and if performance is adequate.
Scenario Description	This scenario requires running through the program execution only once. Begin with test 2.1.5.1.1 as the system is started. All of the 2.1.5 tests, 2.2.1.1, 2.2.1.2, and all of the 2.3.1 tests should be checked during each step of a simple selection process. When the system is doing the summary processing just before displaying the summary information, 2.2.1.3 should be checked. After the summary is displayed, 2.1.4.1.1 and 2.1.4.1.2 should be tested. The database and knowledge base must also be checked.
 Tests	<u>2.1.4.1.1, 2.1.4.1.2, 2.1.5.1.1, 2.1.5.1.2, 2.1.5.1.3, 2.1.5.2.1, 2.1.5.3.5.1,</u> <u>2.1.5.3.5.2, 2.1.5.3.5.3, 2.2.1.1, 2.2.1.2, 2.2.1.3, 2.3.1.2, 2.3.1.3, 2.3.1.4,</u> <u>2.3.1.5, 2.3.1.6</u>
 	

Tests	
Scenario Number	<u>2.4.4</u>
Scenario Objective	To determine if the user selection and summary display functions work correctly when making changes, including changing the order of selections.
Scenario Description	This scenario requires execution of the program all the way through 3 times, with most of the tests checked on each run. The order of the selection process should be mixed up as much as possible, and as differently as possible, on each run. At the end of the first run, the save option should be chosen, and then the selection process should be started over without exiting the program. The retrieve option should be used to bring back the selections saved, and then part of them should be changed before the final summary processing is done. After this second run, the selections should be saved again and then the system should be terminated and restarted. The retrieve option should be used to bring back the selections saved again, and the summary should be processed immediately. Then part of the selections should be changed and the final summary processing should be done again. The database and knowledge base must be checked after each summary is displayed.
 Tests	<u>2.1.3.1.2, 2.1.3.1.3, 2.1.3.2.1, 2.1.3.2.2, 2.1.4.2.1, 2.1.4.2.2, 2.1.4.2.3,</u> <u>2.1.4.3.1, 2.1.5.3.1.1, 2.1.5.3.2.1, 2.1.5.3.3.1, 2.1.5.3.3.4, 2.1.5.3.3.5,</u> <u>2.1.5.3.4.1, 2.1.5.3.4.4, 2.1.5.3.4.5, 2.2.1.3, 2.3.2.1, 2.3.2.2, 2.3.2.3, 2.3.2.4</u>
 	

Tests	
Scenario Number	2.4.5
Scenario Objective	To inspect the code for adherence to structuring constraints.
Scenario Description	This scenario requires no program execution. The code of all subsystems must be checked very carefully for access to the database and the knowledge base and access to the resources of another subsystem.
Tests	2.1.1.3.1, 2.1.2.3.1, 2.3.3.1, 2.3.3.2

Appendix F

Requirements Cross-Reference Matrix

Requirements	
Requirement Number	3.1.A.1
Requirement	Evaluation data collected from outside sources can be entered into the system database. This data can be accepted either interactively or from a file. For data accepted from a file, the system will be able to accept evaluation data from at least two different file formats, at least one of which will be based on the use of keywords in the data file. The system must convert the input into the proper form for the database.
Design Reference	3
Test Reference	2.1.1.1.1, 2.1.1.1.2, 2.4.1
	

Requirements	
Requirement Number	3.1.A.2
Requirement	Evaluation data already in the database may be changed or deleted interactively.
Design Reference	2.2.2
Test Reference	2.1.1.2.1, 2.1.1.2.2, 2.4.1
	

Requirements	
Requirement Number	3.1.A.3
Requirement	All input to the database will be processed by the Knowledge Acquisition Subsystem. Interactive entries will be input via the User Interface Subsystem. The database will be accessed only via the Knowledge Base Subsystem.
Design Reference	2. 2.2. 3
Test Reference	2.1.1.1.1, 2.1.1.1.2, 2.1.1.2.1, 2.1.1.2.2, 2.1.1.3.1, 2.3.3.1, 2.4.1, 2.4.5
	

Requirements	
Requirement Number	3.1.A.4
Requirement	The Knowledge Base Subsystem will provide a mechanism for consistency checking among the entries in its database.
Design Reference	2. 2.2
Test Reference	2.1.1.4.1, 2.4.1
	

Requirements	
Requirement Number	3.1.B.1
Requirement	Knowledge of evaluation features and criteria in the system knowledge base can be added to, changed, or deleted interactively.
Design Reference	2, 2.1, 2.1.1, 2.1.2
Test Reference	2.1.2.1.1, 2.1.2.2.1, 2.1.2.2.2, 2.4.1
  	

Requirements	
Requirement Number	3.1.B.2
Requirement	All interactive input to the knowledge base will be processed by the Knowledge Acquisition Subsystem via the User Interface Subsystem, and the knowledge base will be accessed only via the Knowledge Base Subsystem.
Design Reference	2, 2.1, 2.1.1, 2.1.2, 3
Test Reference	2.1.2.1.1, 2.1.2.2.1, 2.1.2.2.2, 2.1.2.3.1, 2.3.3.1, 2.4.1, 2.4.5
  	

Requirements	
Requirement Number	3.1.B.3
Requirement	The Knowledge Base Subsystem will provide a mechanism for consistency checking among the entries in its knowledge base.
Design Reference	2. 2.1
Test Reference	2.1.2.4.1, 2.4.1
	

Requirements	
Requirement Number	3.1.C.1
Requirement	The user feature and criteria selections can be entered either interactively or from a file saved from a previous session.
Design Reference	1.1, 1.1.1, 1.1.2, 1.2, 1.2.4, 1.2.5, 1.2.6
Test Reference	2.1.3.1.3, 2.3.1.3, 2.4.3, 2.4.4
	

Requirements	
Requirement Number	3.1.C.2
Requirement	If entering feature and criteria selections interactively, the user will have the ability to make changes to any part of the selections at any time, rather than being restricted to entering selections in only one particular order.
Design Reference	1.2, 1.2.4, 1.2.5, 1.2.6, 1.6
Test Reference	2.1.3.1.3, 2.1.3.2.1, 2.1.3.2.2, 2.4.4
	

Requirements	
Requirement Number	3.1.C.3
Requirement	After entering feature and criteria selections from a file, the user will be able to make interactive modifications to these selections just as if they had been entered interactively in the first place.
Design Reference	1.2, 1.2.4, 1.2.5, 1.2.6
Test Reference	2.1.3.1.3, 2.4.4
	

Requirements	
Requirement Number	3.1.C.4
Requirement	Input will be verified by the system as it is entered. If the input is not appropriate, it will be discarded and new input will be requested. Appropriate informative error messages will be provided.
Design Reference	<u>1.2, 1.2.4, 1.2.5, 1.2.6</u>
Test Reference	<u>2.1.3.1.1, 2.3.2.1, 2.3.2.2, 2.3.2.3, 2.4.2, 2.4.4</u>
  	

Requirements	
Requirement Number	3.1.D.1
Requirement	The selection features and criteria can be saved to a file at any time during a session by a single user action. This file can then be used in a subsequent session.
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.2</u>
Test Reference	<u>2.1.3.1.2, 2.1.4.3.1, 2.4.4</u>
  	

Requirements	
Requirement Number	3.1.D.2
Requirement	At any time during a session, the user can cause the information in the current window to be sent to the printer. An option will be available to the user to print the information exactly as it looks in the window or else to print a report which clearly provides all or some portion of the information which is displayed in the window.
Design Reference	1.2, 1.2.6, 1.4, 1.4.3
Test Reference	2.1.4.1.1, 2.1.4.1.2, 2.4.3
	

Requirements	
Requirement Number	3.1.D.3
Requirement	After the user has completely specified all required selection features and criteria, the system will provide the user with windows of evaluation recommendations, information, and analysis at multiple levels of detail. The first (highest) level of detail will provide recommendations to the user, while the second level of detail will provide analysis information. This analysis information will include an explanation of calculations used to arrive at the recommendations, and it will provide the user with the ability to change parameters used in these calculations. Further levels will provide progressively more detailed information until the lowest level will provide the user with the evaluation data from the database.
Design Reference	1.2, 1.2.6, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.5.4
Test Reference	2.1.4.2.1, 2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.4.2, 2.4.4
	

Requirements	
Requirement Number	3.1.E.1
Requirement	Any icons or other conventions used will be clearly explained to the user at the beginning of a session. Optionally, the information will be available in a help window, via one mouse click or one keystroke, to the user who wants to see such explanations. In any case, the display of the help window containing such explanations will be an option available to the user from any window.
Design Reference	1.2, 1.2.6
Test Reference	2.1.5.1.1, 2.1.5.1.2, 2.3.1.2, 2.4.3
	

Requirements	
Requirement Number	3.1.E.2
Requirement	The active window will clearly invite one or more of the following responses from the user: (a) One or more choices, as appropriate, may be selected by the click of a mouse (or equivalent) in a clearly marked area of the screen which is associated with a (clear) description of the choice. (b) An action may be selected by the click of a mouse (or equivalent) in a clearly marked area of the screen which is associated with a (clear) description of the action. (c) Text may be input from the keyboard into a field which is highlighted on the screen and/or where the cursor is located.
Design Reference	1.2, 1.2.6, 4
Test Reference	2.3.1.3, 2.3.1.6, 2.4.3
	

Requirements	
Requirement Number	3.1.E.3
Requirement	The system will clearly lead the decision maker through the process of selecting features and criteria in preparation for the summary of recommendations and information to be provided to the decision maker. This process must be complete before the summary can be provided.
Design Reference	1.2, 1.2.1, 1.2.6
Test Reference	2.1.4.2.1, 2.3.1.2, 2.3.1.3, 2.4.3, 2.4.4
	

Requirements	
Requirement Number	3.1.E.4
Requirement	The system will determine from the decision maker the type of software about which to make recommendations. The software type may be specified as a particular individual tool, a particular tool set, the software which will support a particular life cycle activity, or an entire APSE.
Design Reference	1.2, 1.2.2, 1.2.6
Test Reference	2.1.5.3.1.1, 2.4.2
	

Requirements	
Requirement Number	3.1.E.5
Requirement	The application area for the software will be specified by the decision maker. This information will be used to assist the decision maker by suggesting features and criteria which are usually important in making software selections for the particular application area.
Design Reference	1.2, 1.2.3, 1.2.6
Test Reference	2.1.5.3.2.1, 2.1.5.3.3.1, 2.1.5.3.4.1, 2.4.2
	

Requirements	
Requirement Number	3.1.E.6
Requirement	The system will provide the decision maker with suggested features for selection. Other features may be added by the user, but they must be consistent with the features in the knowledge base. The system will also provide the capability for using multiple levels of detail for feature specification. At the finest level of detail for any given feature, the user will be able to specify that feature to the finest level of detail consistent with the knowledge in the system knowledge base for handling that feature. The system will provide default weights for all features selected, and the user will also be able to change those weights.
Design Reference	1.2, 1.2.4, 1.2.6
Test Reference	2.1.5.3.3.2, 2.1.5.3.3.3, 2.1.5.3.3.4, 2.1.5.3.3.5, 2.4.2, 2.4.4
	

Requirements	
Requirement Number	3.1.E.7
Requirement	The system will provide the decision maker with suggested criteria for selection. Other criteria may be added by the user, but they must be consistent with the criteria in the knowledge base. The system will also provide the capability for using multiple levels of detail for criteria specification. At the finest level of detail for any given criterion, the user will be able to specify that criterion to the finest level of detail consistent with the knowledge in the system knowledge base for handling that criterion. The system will provide default weights for all criteria selected, and the user will also be able to change those weights.
Design Reference	1.2, 1.2.5, 1.2.6
Test Reference	2.1.5.3.4.2, 2.1.5.3.4.3, 2.1.5.3.4.4, 2.1.5.3.4.5, 2.4.2, 2.4.4

Requirements	
Requirement Number	3.1.E.8
Requirement	When making selections, the decision maker will usually not be required to enter words using the keyboard. Choices will be provided which will require either a click of a mouse or one keystroke to select, except in cases where this is not practical. Selection choices will not be implemented so as to restrict the decision maker from having flexibility in making choices. However, it will ensure that selections always be valid and consistent with the current system knowledge base.
Design Reference	1.2, 1.2.4, 1.2.5, 1.2.6
Test Reference	2.1.5.3.3.4, 2.1.5.3.4.4, 2.1.5.3.5.1, 2.1.5.3.5.2, 2.1.5.3.5.3, 2.4.3, 2.4.4

Requirements	
Requirement Number	3.1.E.9
Requirement	The system will provide a summary of recommendations and other information to the decision maker after the selection of features and criteria has been completed. All recommendations and information provided will be based on the evaluation information data in the database, and this data will be treated in accordance with the feature and criteria knowledge provided in the knowledge base.
Design Reference	1.2, 1.2.6, 1.5, 1.5.1, 1.5.2, 1.5.4, 2.1.3, 2.1.3.1, 2.1.3.2, 2.1.4, 2.1.4.1, 2.1.4.2, 2.1.4.3, 2.1.4.4, 2.1.4.5
Test Reference	2.1.4.2.1, 2.1.5.3.6.1, 2.4.2, 2.4.4
	

Requirements	
Requirement Number	3.1.E.10
Requirement	The system will provide the decision maker with multiple levels of detail for viewing the summary of recommendations and other information. At the finest level of detail, the user will be able to review the actual data in the system database.
Design Reference	1.2, 1.2.6, 1.5, 1.5.4
Test Reference	2.1.4.2.5, 2.4.2
	

Requirements	
Requirement Number	3.1.E.11
Requirement	A form of graphical browser will be available for the user to view at any time to maintain a perspective of the organization of the system processing, as well as how the current activity fits into this processing.
Design Reference	1.2, 1.2.6, 1.4, 1.4.5
Test Reference	2.1.5.1.3, 2.4.3
	

Requirements	
Requirement Number	3.1.E.12
Requirement	At any time during a session, the user can cause the system to display a summary of the current selections and/or the reasoning used to get the summary of recommendations and information resulting from the current selections.
Design Reference	1.2, 1.2.6, 1.3, 1.4, 1.4.4
Test Reference	2.1.5.2.1, 2.4.3
	

Requirements	
Requirement Number	3.1.E.13
Requirement	During periods of delay, when the system is processing before putting a new display on the screen, the system will always provide the user with a clear indication that the system is working.
Design Reference	<u>1.2, 1.2.6, 1.5.4</u>
Test Reference	<u>2.3.1.4, 2.4.3</u>
	

Requirements	
Requirement Number	3.1.E.14
Requirement	The user will be able to quit from the system at any time. However, a reminder will always be given that the user may want to save the current selections first (if that has not already been done).
Design Reference	<u>1.2, 1.2.6, 1.4</u>
Test Reference	<u>2.3.1.5, 2.4.3</u>
	

Requirements	
Requirement Number	3.1.E.15
Requirement	At any time that a user request cannot be satisfied or an error condition occurs, an appropriate informative message will be provided to the user, and regular program execution will continue.
Design Reference	<u>1.2, 1.2.6</u>
Test Reference	<u>2.3.2.2, 2.3.2.3, 2.3.2.4, 2.4.4</u>
	

Requirements	
Requirement Number	3.1.E.16
Requirement	On-line help will always be available, with the click of a mouse or one keystroke, to give more complete explanations for the options possible from the current window.
Design Reference	<u>1.2, 1.2.6, 1.4, 1.4.1</u>
Test Reference	<u>2.1.5.1.2, 2.4.3</u>
	

Requirements	
Requirement Number	3.2.1
Requirement	The system must have a response time of less than 5 seconds from the time of any interactive input causing a new window to be displayed until that new window is displayed on the screen.
Design Reference	1.2, 1.2.6
Test Reference	2.2.1.1, 2.2.1.3, 2.4.3
	

Requirements	
Requirement Number	3.2.2
Requirement	A window must be displayed in its entirety, from first pixel to last, in less than 2 seconds.
Design Reference	1.5.3
Test Reference	2.2.1.2, 2.4.3
	

Requirements	
Requirement Number	3.2.3
Requirement	The average response time for each interactive input, except for the processing which displays summary recommendations, will be less than 5 seconds.
Design Reference	1.2, 1.2.6
Test Reference	2.2.1.1, 2.4.3
	

Requirements	
Requirement Number	3.2.4
Requirement	The full response time for the processing which displays summary recommendations will be expected to vary greatly depending upon the amount of data in the database. However, the maximum delay time between displays of at least some portion of the summary output will be 30 seconds, and the total delay time will not exceed 1 second for every evaluation in the database which is processed for the type of software specified by the user.
Design Reference	1.2, 1.2.6, 1.5, 1.5.4
Test Reference	2.2.1.3, 2.4.3
	

Requirements	
Requirement Number	3.2.5
Requirement	The size of the database and knowledge base used by the system will only be restricted by the size of the storage area available. It will not be artificially restricted by the system.
Design Reference	2
Test Reference	2.2.2.1, 2.4.1
	

Requirements	
Requirement Number	3.3.1
Requirement	The knowledge of what the system should do with the evaluation data in the database is contained in the knowledge base. The knowledge base contains the evaluation features and criteria to be used as well as the specifics of how they should be used. This means that changes to the knowledge base will change how the system will deal with the information in the database.
Design Reference	2.1, 2.1.4, 2.1.4.1, 2.1.4.2, 2.1.4.3, 2.1.4.4, 2.1.4.5
Test Reference	2.1.5.3.6.1, 2.4.2
	

Requirements	
Requirement Number	3.3.2
Requirement	The main system logic in the Decision Logic Subsystem will access the algorithms which use the knowledge in the knowledge base to determine how to deal with the data in the database. However, the logic will work based only on the form of the knowledge, not the content.
Design Reference	<u>1, 1.2, 1.2.6</u>
Test Reference	<u>2.1.5.3.6.1, 2.3.3.1, 2.4.2, 2.4.5</u>
	

Requirements	
Requirement Number	3.3.3
Requirement	Each of the subsystems will be independent of each other's implementation. Each will depend only on explicitly defined interfaces to use another's resources.
Design Reference	<u>0, 1, 2, 3, 4</u>
Test Reference	<u>2.3.3.1, 2.3.3.2, 2.4.5</u>
	

Requirements	
Requirement Number	3.4.1
Requirement	The expected system user is the decision maker. The system will provide a mechanism for a system manager to use the Knowledge Acquisition Subsystem to modify the database and/or the knowledge base, but this mechanism will not be visible to the decision maker.
Design Reference	3
Test Reference	2.1.1.5.1, 2.1.2.5.1, 2.4.1
	

Requirements	
Requirement Number	3.4.2
Requirement	The system will always present an active window to the user. One window will not disappear without another immediately appearing.
Design Reference	1.5, 1.5.3, 4
Test Reference	2.3.1.4, 2.3.1.6, 2.4.3
	

Requirements	
Requirement Number	3.4.3
Requirement	If other windows in addition to the active window are on the screen, the active window will always be clearly marked and fully visible.
Design Reference	<u>1.5, 1.5.3, 4</u>
Test Reference	<u>2.3.1.4, 2.3.1.6, 2.4.3</u>
	

Requirements	
Requirement Number	3.4.4
Requirement	Screen windows will be able to provide graphics and/or text.
Design Reference	<u>1.5, 1.5.3, 4</u>
Test Reference	<u>2.3.1.6, 2.4.3</u>
	

Requirements	
Requirement Number	3.4.5
Requirement	The user will not be restricted to "backing out" from a particular level of processing (such as when viewing lower levels of detail) by retracing steps through every window that led there. The system must (at least appear to) employ the hypertext concept of nonlinear organization of chunks of information.
Design Reference	1.2, 1.2.6, 1.4, 1.4.6
Test Reference	2.3.1.3, 2.4.3
	

Requirements	
Requirement Number	3.4.6
Requirement	The system must run on a contemporary workstation which provides both graphics and textual capabilities.
Design Reference	4
Test Reference	2.3.1.1, 2.4.2
	

Appendix G

Design Documentation for ASSIST

ASSIST by Pat Lawlis**Design Documentation**

Number	Module Name	Number	Module Name
0	ASSIST	1.5.1	Show Parameters
1	Decision Logic Subsystem	1.5.2	Show Missing Info
1.1	Choosing	1.5.3	Display Resources
1.1.1	Features	1.5.4	Display Summary
1.1.2	Criteria	2	Knowledge Base Subsystem
1.2	Process	2.1	Knowledge Base
1.2.1	Initialize Data	2.1.1	Store Knowledge
1.2.2	Get Context	2.1.2	Retrieve Knowledge
1.2.3	Get Application Area	2.1.3	Keep Summary
1.2.4	Get Features	2.1.3.1	Overall Summaries
1.2.5	Get Criteria	2.1.3.2	DB Data Summary
1.2.6	Control	2.1.4	Interpret Knowledge
1.3	Choices	2.1.4.1	Find Pertinent Data
1.4	Support	2.1.4.2	Summarize Data
1.4.1	Help	2.1.4.3	Summarize Individual Features and Criteria
1.4.2	Save or Retrieve	2.1.4.4	Summarize Implementation IDs
1.4.3	Print	2.1.4.5	Give Rating
1.4.4	Review	2.2	Database
1.4.5	Browse	3	Knowledge Acquisition Subsystem
1.4.6	Backtrack	4	User Interface Subsystem
1.5	Display Info		

Design Documentation ASSIST Diagram 	Description ASSIST will use a collection of evaluation data and user-given parameters in preparing and presenting advice to the user concerning APSE and other Ada software selection decisions.			
	Design Decisions The ASSIST subsystems will follow the general form of a DSS. The subsystems will be: User Interface, Decision Logic, Knowledge Acquisition, and Knowledge Base.			
	<table border="1"> <tr> <td> Requirements Design Level 0 </td> <td> 3.3.3   </td> <td> Tests 2.3.3.1, 2.3.3.2, 2.4.5   </td> </tr> </table>	Requirements Design Level 0	3.3.3  	Tests 2.3.3.1, 2.3.3.2, 2.4.5  
	Requirements Design Level 0	3.3.3  	Tests 2.3.3.1, 2.3.3.2, 2.4.5  	

Description

All information is input to the system through the Knowledge Acquisition Subsystem. The system manager is a special user who adds evaluation data to the database and adds knowledge to the knowledge base about how to deal with the application area, features, and criteria associated with the type of software to be evaluated. The decision maker is the regular user who chooses the features and criteria to be used for a particular session. These choices may be saved for later reuse and then recalled. The Decision Logic Subsystem directs the system processing. The knowledge base massages the data from the database, and from this ASSIST provides the user with requested evaluation information at varying levels of abstraction. The user may request a review of the choices made or request a printed report of the information given and/or reasoning used at any time during the session.

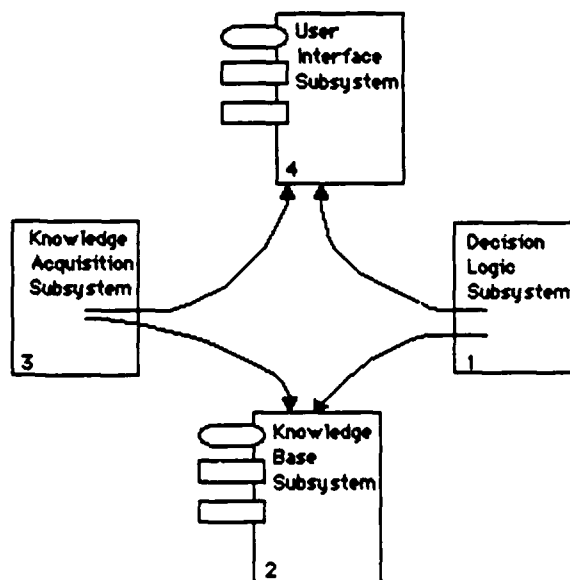


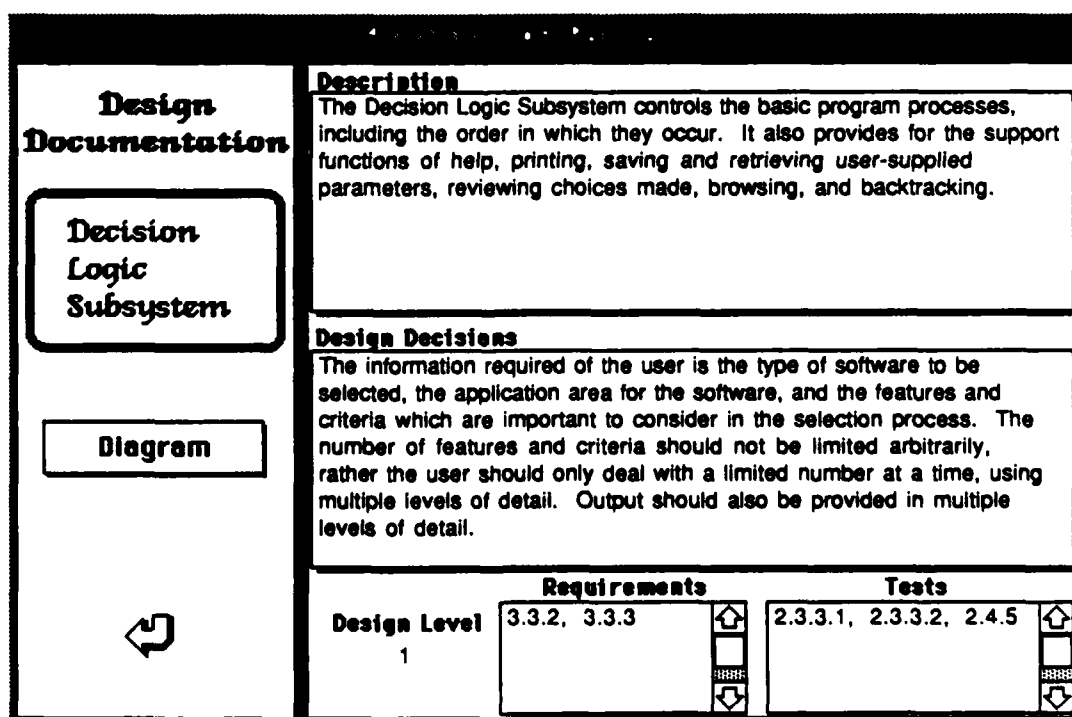
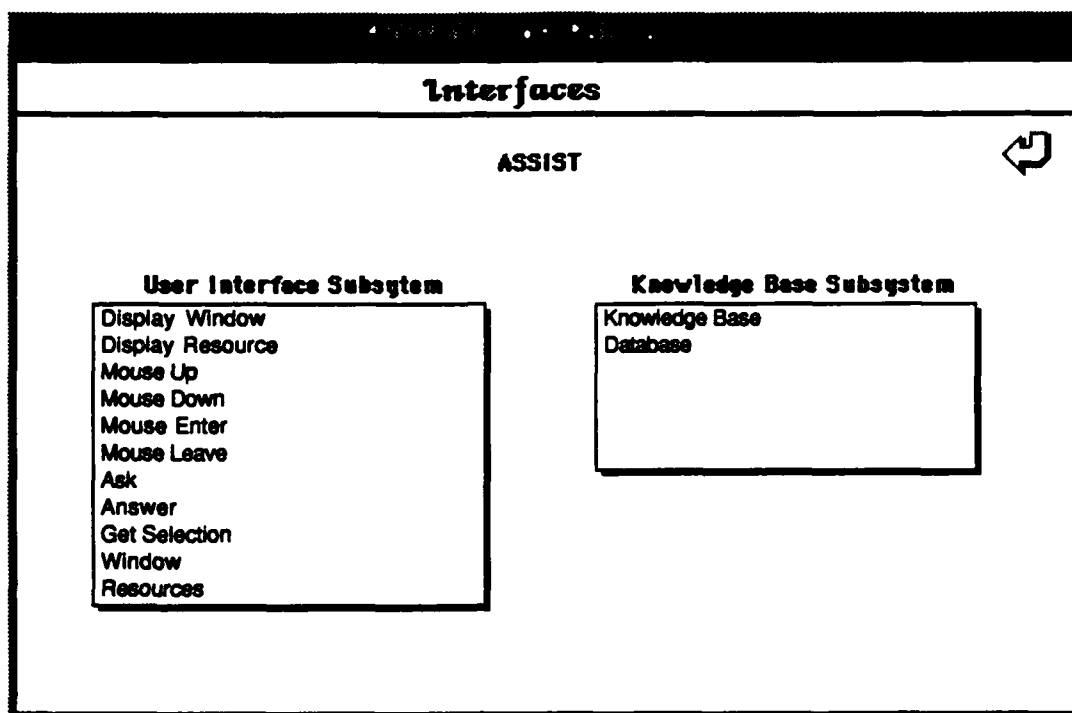
Design Diagram

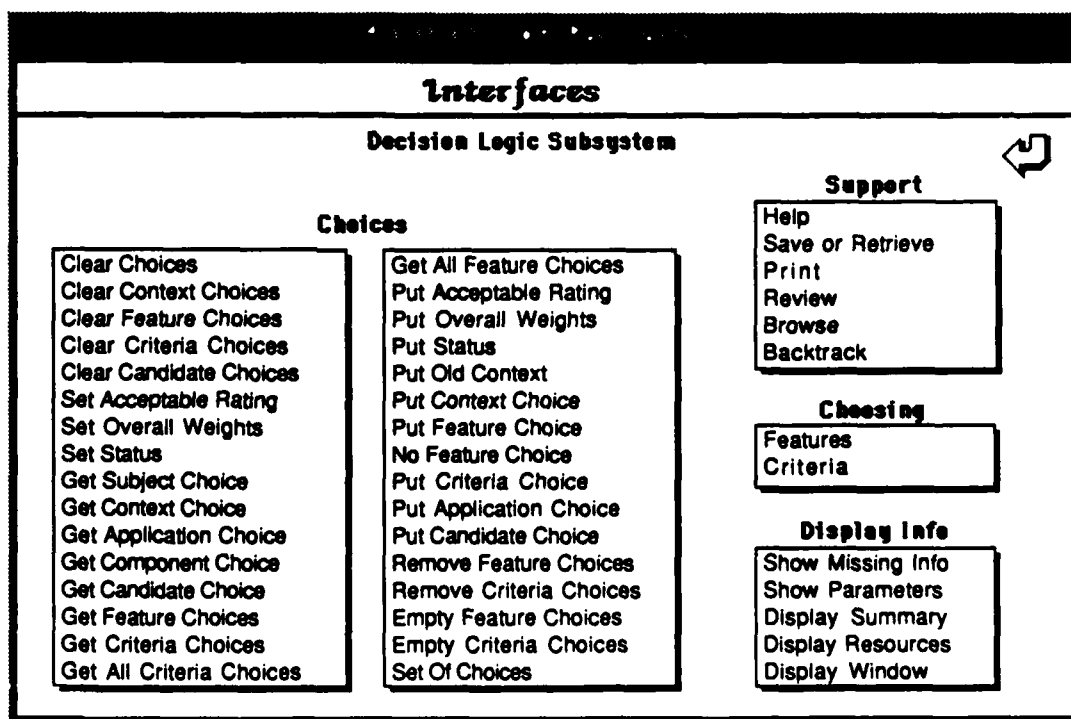
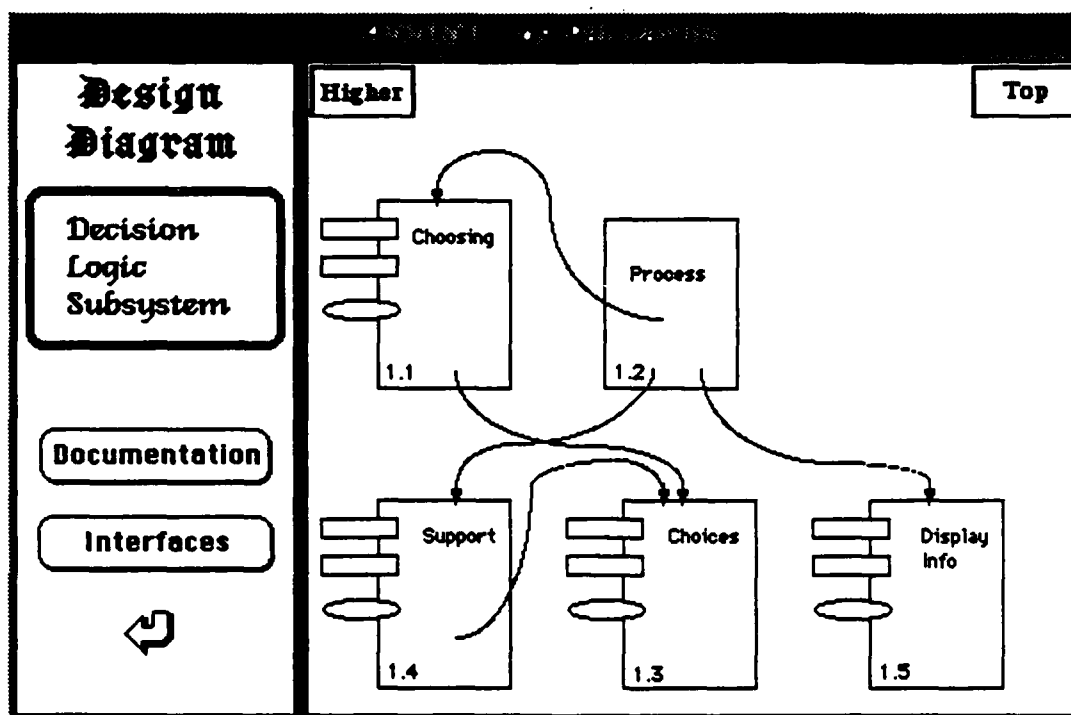
ASSIST

Documentation

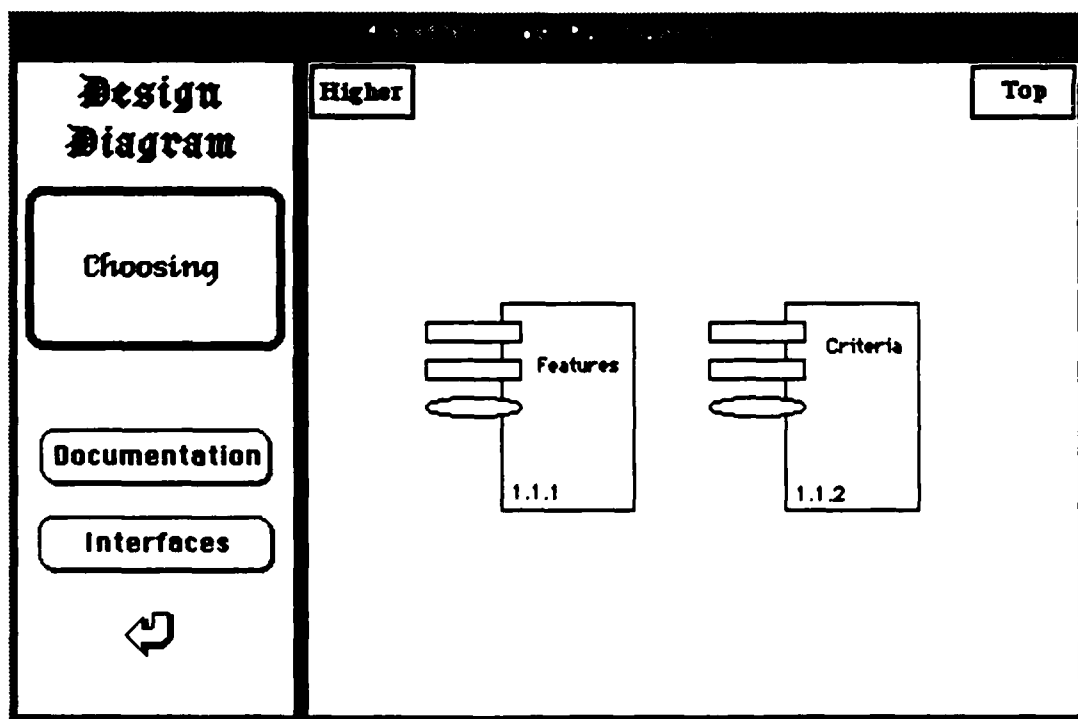
Interfaces







Design Documentation									
Choosing Diagram ↩	Description This package consists of the resources which direct the acquisition of choosing the features and criteria to be used in selecting the software. This includes acquiring the weights for each of the choices.								
	Design Decisions This package consists of two parallel packages, one for feature resources and one for criteria resources. The functions performed by each are very similar, and a generic package could be used for many of the resources.								
	<table border="1"> <thead> <tr> <th></th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>Design Level</td> <td>3.1.C.1</td> <td>2.3.1.3, 2.4.3</td> </tr> <tr> <td>1.1</td> <td></td> <td></td> </tr> </tbody> </table>		Requirements	Tests	Design Level	3.1.C.1	2.3.1.3, 2.4.3	1.1	
	Requirements	Tests							
Design Level	3.1.C.1	2.3.1.3, 2.4.3							
1.1									

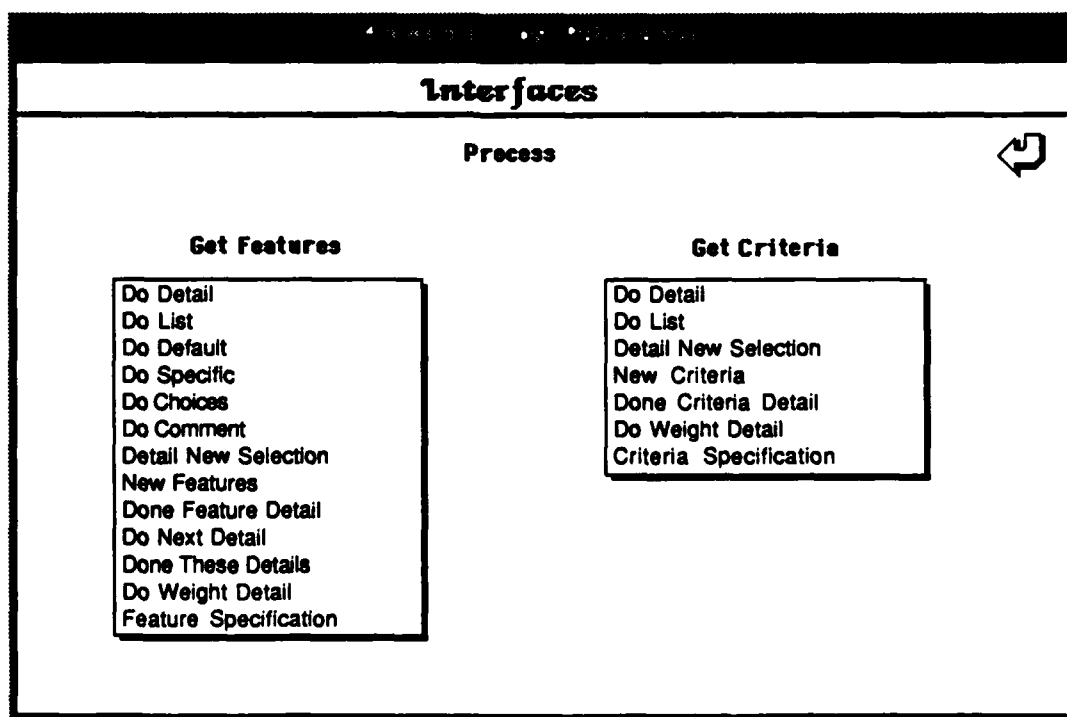
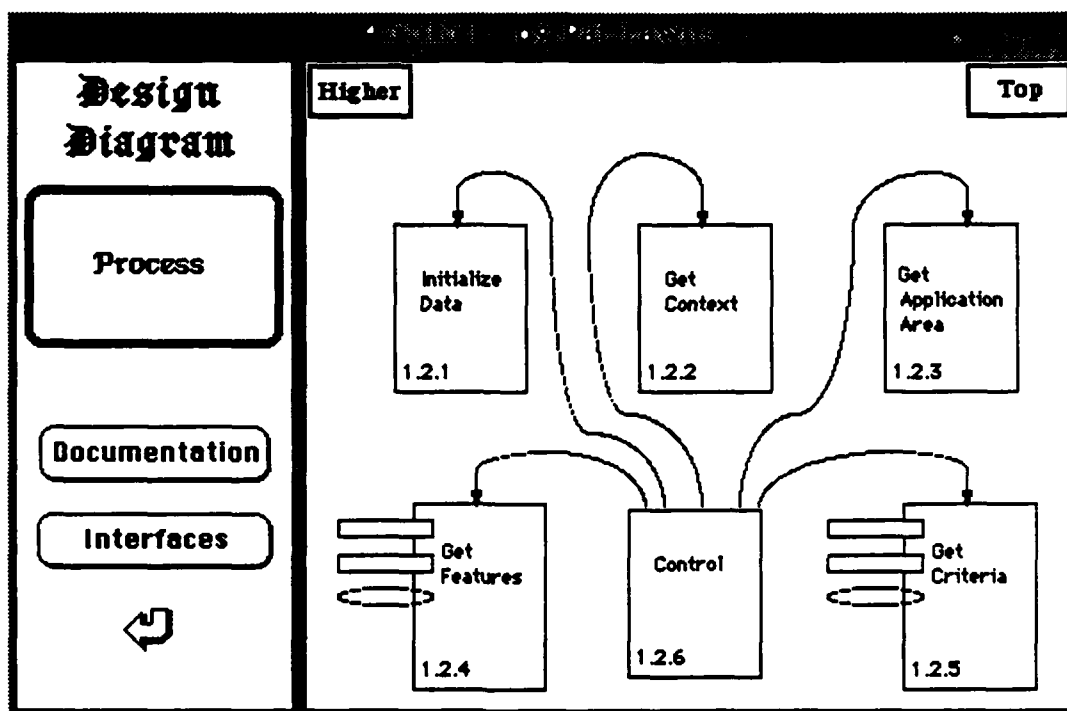


Interfaces			
	Display Features	Choosing	Display Criteria
	No Feature Choices		No Criteria Choices
	UnChoose Feature		UnChoose Criterion
	UnSelect Top Feature		UnSelect Top Criterion
F	Restore Old Selections	C	Restore Old Selections
•	Put Specific Value	r	Put Specific Value
•	Get Default Features	i	Set Up Default Criteria
t	Ask About Details	t	Ask About Details
■	Display New Features	•	Display New Criterion
r	Add To Displayed List	r	Add To Displayed List
•	Replace Feature	i	Replace Criterion
s	Show Possible Adds	•	Show Possible Adds
	Get Visible Features		Get Visible Criteria
	Show Feature Choices		Show Criteria Choices
	Replace Or Forget		Replace Or Forget
	UnSelect Detail Feature		Clear Weight Criteria
	Get Feature Choices		Get Criteria Choices
	Put Feature Choices		Put Criteria Choices
	Top Level Feature Window		Top Level Criteria Window
	Detail Feature Window		Detail Criteria Window
	Top Level Weight Window		Top Level Weight Window
	Detail Weight Window		Detail Weight Window

Design Documentation Features ↩	Description This package contains the resources for interactively acquiring the features chosen as well as the corresponding weight for each feature.		
	Design Decisions Each level of detail will be displayed similarly. The only input which will be typed by the user is numerical input, and it will be validated. Other types of input will be chosen from lists.		
	Design Level 1.1.1	Requirements 3.1.C.1	Tests 2.3.1.3, 2.4.3

Design Documentation		Description	
Criteria		This package contains the resources for interactively acquiring the criteria chosen as well as the corresponding weight for each criterion.	
		Design Decisions Each level of detail will be displayed similarly. The only input which will be typed by the user is numerical input, and it will be validated. Other types of input will be chosen from lists.	
		Requirements Design Level 1.1.2	Tests 2.3.1.3, 2.4.3

Design Documentation		Description	
Process Diagram PDL		This subprogram controls the process of choosing the type of software to be selected, the application area, and the features and criteria to be used. It ensures that all dependencies are handled if the user decides to make choices out of the ordinary sequence. When it has all the necessary choices, it sets the process in motion to use these choices to produce recommendations and other information for the use of the decision maker.	
		Design Decisions The user will be permitted to make choices out of the ordinary sequence provided by Process. This requires the system to gracefully handle dependencies among the choices, providing gentle reminders as necessary.	
	Requirements Design Level 1.2	Tests 2.1.3.1.1, 2.1.3.1.2, 2.1.3.1.3, 2.1.3.2.1, 2.1.3.2.2, 2.1.4.1.1, 2.1.4.1.2, 2.1.4.2.1,	



Program Design Language (PDL)

Subprogram Process

initialize data
 get software context
 get application area
 get top level features
 get detail features
 get top level feature weights
 get detail feature weights
 get top level criteria
 get detail criteria
 get top level criteria weights
 get detail criteria weights



Design Documentation

Initialize Data



Description
 This subprogram initializes all of the data to be used in displaying software characteristics and making choices.

Design Decisions
 System parameters are set to default values.

	Requirements	Tests
Design Level 1.2.1	3.1.E.3  	2.3.1.2, 2.3.1.3, 2.4.3  

Design Documentation		Description	
Get Context	This subprogram directs the process of getting the user to specify the type of software to be selected.		
	Design Decisions Choices will be made from lists.		
	Design Level 1.2.2	Requirements 3.1.E.4	Tests 2.1.5.3.1.1, 2.4.4

Design Documentation		Description	
Get Application Area	This subprogram directs the process of getting the user to specify the application area of software to be selected.		
	Design Decisions Choice will be made from a list which will include one general choice.		
	Design Level 1.2.3	Requirements 3.1.E.5	Tests 2.1.5.3.2.1, 2.4.4

Design Documentation Get Features 	Description This package provides the resources for the process of getting the user to specify the important features of the software to be selected, along with their associated weights. This includes working with the features at multiple levels of detail.											
	Design Decisions The number of choices available at any one time should be within the manageable range of complexity of 7-or-2. The lists of possible features come from the knowledge base. Any top level feature will be available for choosing, regardless of application area, but the application area will determine the suggested features. Some detail features will be general and others will apply to specific types of software. Specification of the highest feature weight will mean that the feature must be present in the implementation in order for it to be considered by ASSIST.											
	Design Level 1.2.4	<table border="1"> <thead> <tr> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>3.1.C.1, 3.1.C.2, </td> <td>2.1.3.1.1, 2.1.3.1.3, </td> </tr> <tr> <td>3.1.C.3, 3.1.C.4, </td> <td>2.1.3.2.1, 2.1.3.2.2, </td> </tr> <tr> <td>3.1.E.6, 3.1.E.8 </td> <td>2.1.5.3.3.2, 2.1.5.3.3.3, </td> </tr> <tr> <td></td> <td>2.1.5.3.3.4, 2.1.5.3.3.5, </td> </tr> </tbody> </table>	Requirements	Tests	3.1.C.1, 3.1.C.2, 	2.1.3.1.1, 2.1.3.1.3, 	3.1.C.3, 3.1.C.4, 	2.1.3.2.1, 2.1.3.2.2, 	3.1.E.6, 3.1.E.8 	2.1.5.3.3.2, 2.1.5.3.3.3, 		2.1.5.3.3.4, 2.1.5.3.3.5, 
	Requirements	Tests										
3.1.C.1, 3.1.C.2, 	2.1.3.1.1, 2.1.3.1.3, 											
3.1.C.3, 3.1.C.4, 	2.1.3.2.1, 2.1.3.2.2, 											
3.1.E.6, 3.1.E.8 	2.1.5.3.3.2, 2.1.5.3.3.3, 											
	2.1.5.3.3.4, 2.1.5.3.3.5, 											

Design Documentation Get Criteria 	Description This package provides the resources for the process of getting the user to specify the important criteria of the software to be selected, along with their associated weights. This includes working with the criteria at multiple levels of detail.											
	Design Decisions The number of choices available at any one time should be within the manageable range of complexity of 7-or-2. The lists of possible criteria come from the knowledge base. Any criterion will be available for choosing, regardless of application area, but the application area will determine the suggested criteria.											
	Design Level 1.2.5	<table border="1"> <thead> <tr> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>3.1.C.1, 3.1.C.2, </td> <td>2.1.3.1.1, 2.1.3.1.3, </td> </tr> <tr> <td>3.1.C.3, 3.1.C.4, </td> <td>2.1.3.2.1, 2.1.3.2.2, </td> </tr> <tr> <td>3.1.E.7, 3.1.E.8 </td> <td>2.1.5.3.4.2, 2.1.5.3.4.3, </td> </tr> <tr> <td></td> <td>2.1.5.3.4.4, 2.1.5.3.4.5, </td> </tr> </tbody> </table>	Requirements	Tests	3.1.C.1, 3.1.C.2, 	2.1.3.1.1, 2.1.3.1.3, 	3.1.C.3, 3.1.C.4, 	2.1.3.2.1, 2.1.3.2.2, 	3.1.E.7, 3.1.E.8 	2.1.5.3.4.2, 2.1.5.3.4.3, 		2.1.5.3.4.4, 2.1.5.3.4.5, 
	Requirements	Tests										
3.1.C.1, 3.1.C.2, 	2.1.3.1.1, 2.1.3.1.3, 											
3.1.C.3, 3.1.C.4, 	2.1.3.2.1, 2.1.3.2.2, 											
3.1.E.7, 3.1.E.8 	2.1.5.3.4.2, 2.1.5.3.4.3, 											
	2.1.5.3.4.4, 2.1.5.3.4.5, 											

Design Documentation		Description																
Control ↶	This is the control for subprogram Process. It uses the resources of Process as necessary.																	
	Design Decisions Control will normally direct the user to specify features and criteria in order. However, it will permit specifications to be made out of the usual order as long as system integrity may be maintained.																	
	Design Level 1.2.6	Requirements <table border="1"> <tr> <td>3.1.C.1, 3.1.C.2,</td> <td>↑</td> </tr> <tr> <td>3.1.C.3, 3.1.C.4,</td> <td>□</td> </tr> <tr> <td>3.1.D.1, 3.1.D.2,</td> <td>▨</td> </tr> <tr> <td>3.1.D.3, 3.1.E.1,</td> <td>↓</td> </tr> </table>	3.1.C.1, 3.1.C.2,	↑	3.1.C.3, 3.1.C.4,	□	3.1.D.1, 3.1.D.2,	▨	3.1.D.3, 3.1.E.1,	↓	Tests <table border="1"> <tr> <td>2.1.3.1.1, 2.1.3.1.2,</td> <td>↑</td> </tr> <tr> <td>2.1.3.1.3, 2.1.3.2.1,</td> <td>□</td> </tr> <tr> <td>2.1.3.2.2, 2.1.4.1.1,</td> <td>▨</td> </tr> <tr> <td>2.1.4.1.2, 2.1.4.2.1,</td> <td>↓</td> </tr> </table>	2.1.3.1.1, 2.1.3.1.2,	↑	2.1.3.1.3, 2.1.3.2.1,	□	2.1.3.2.2, 2.1.4.1.1,	▨	2.1.4.1.2, 2.1.4.2.1,
3.1.C.1, 3.1.C.2,	↑																	
3.1.C.3, 3.1.C.4,	□																	
3.1.D.1, 3.1.D.2,	▨																	
3.1.D.3, 3.1.E.1,	↓																	
2.1.3.1.1, 2.1.3.1.2,	↑																	
2.1.3.1.3, 2.1.3.2.1,	□																	
2.1.3.2.2, 2.1.4.1.1,	▨																	
2.1.4.1.2, 2.1.4.2.1,	↓																	

Design Documentation		Description																
Choices ↶	This package contains all the resources for internally saving and retrieving the choices which have been made by the decision maker about each stage of the software selection process.																	
	Design Decisions Features and criteria are saved along with their respective weights. This expedites the summary calculations.																	
	Design Level 1.3	Requirements <table border="1"> <tr> <td>3.1.C.2, 3.1.E.12</td> <td>↑</td> </tr> <tr> <td></td> <td>□</td> </tr> <tr> <td></td> <td>▨</td> </tr> <tr> <td></td> <td>↓</td> </tr> </table>	3.1.C.2, 3.1.E.12	↑		□		▨		↓	Tests <table border="1"> <tr> <td>2.1.3.2.1, 2.1.3.2.2,</td> <td>↑</td> </tr> <tr> <td>2.4.4</td> <td>□</td> </tr> <tr> <td></td> <td>▨</td> </tr> <tr> <td></td> <td>↓</td> </tr> </table>	2.1.3.2.1, 2.1.3.2.2,	↑	2.4.4	□		▨	
3.1.C.2, 3.1.E.12	↑																	
	□																	
	▨																	
	↓																	
2.1.3.2.1, 2.1.3.2.2,	↑																	
2.4.4	□																	
	▨																	
	↓																	

Design Documentation		Description	
Support Diagram ↩	This package contains the resources for all of the support functions which are available to the decision maker at all times during the execution of ASSIST.		
	Design Decisions The support functions are: a help facility, a capability to save and retrieve choices made by the decision maker, a capability to print information from ASSIST, a capability to review the choices which have been made, a graphical browsing capability, and a backtracking capability.		
	Design Level 1.4	Requirements 3.1.D.1, 3.1.D.2, 3.1.E.11, 3.1.E.12, 3.1.E.14, 3.1.E.16, 3.4.5	Tests 2.1.3.1.2, 2.1.4.1.1, 2.1.4.1.2, 2.1.4.3.1, 2.1.5.1.2, 2.1.5.1.3, 2.1.5.2.1, 2.3.1.3,

Design Diagram		Higher		Top	
Support Documentation Interfaces ↩	Help 1.4.1 Save or Retrieve 1.4.2 Print 1.4.3 Review 1.4.4 Browse 1.4.5 Backtrack 1.4.6				

Interfaces

Support 

Help	Save or Retrieve	Print
Show Help Go Other Help Help Window	Save Retrieve Save Structure	Print Window Print Report Window

Review	Browse
Put Subject Header Put Application Header Put Feature Header Put Criteria Header Put Parameters Header Put Other Info Review Window	Show Location Go Location Browse Window

Design Documentation

Help



Description

This package provides the resources for accessing the appropriate part of the system on-line help.

Design Decisions

When help is requested by the user, specific help for the current activity will be displayed, with easy access to other help as well. Help facilities will appear hypertext-like to the user. Help for an activity will include the purpose of the activity, actions which are a part of the activity, any constraints to these actions, and anything which may appear unusual about the activity.

	Requirements	Tests
Design Level 1.4.1	3.1.E.16 	2.1.5.1.2, 2.4.3

Design Documentation		Description	
Save or Retrieve	This package provides the resources for saving user-specified choices to disk, and also for retrieving choices which have been saved during a previous session with ASSIST.		
	Design Decisions The user should be able to specify a specific file name or use a default name. Retrieved choices will replace any prior specifications. Even if what was saved was incomplete, all prior specifications will be removed.		
	Design Level 1.4.2	Requirements 3.1.D.1	Tests 2.1.3.1.2, 2.1.4.3.1, 2.4.3

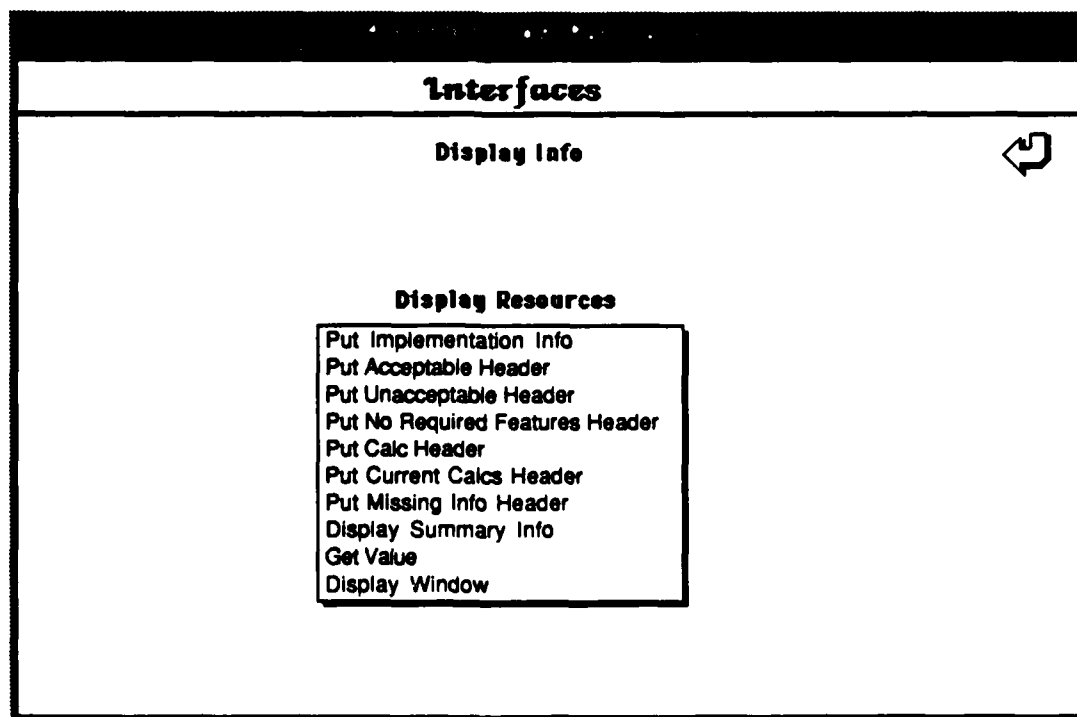
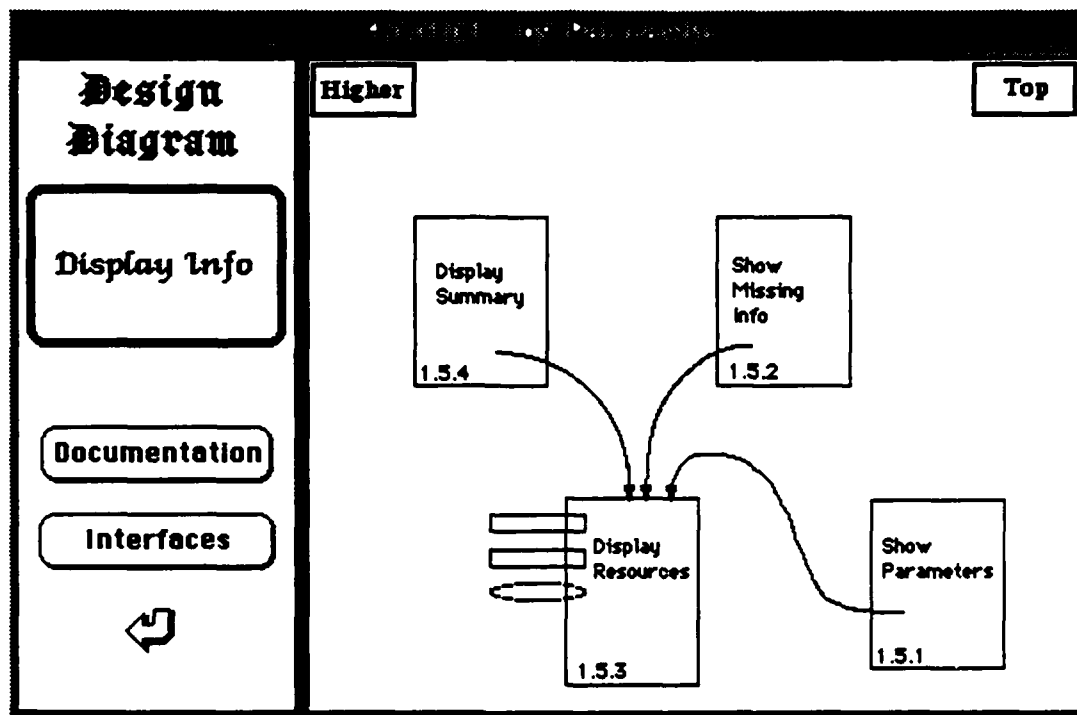
Design Documentation		Description	
Print	This package provides the resources for printing information from ASSIST. These resources include the ability to print the information in a window and to print reports concerning the system processing.		
	Design Decisions Reports will be able to show: parameters chosen by the user, recommendations provided by ASSIST, system parameters, and any other analyses provided by ASSIST.		
	Design Level 1.4.3	Requirements 3.1.D.2	Tests 2.1.4.1.1, 2.1.4.1.2, 2.4.3

Design Documentation				
Review	Description This package contains the resources for providing the decision maker with a review of the decisions already made and a perspective on those which still need to be made.			
	Design Decisions The results (recommendations and analyses) provided by ASSIST do not need to be repeated by this activity, but all user choices and system parameters must be provided.			
	<table border="1"> <thead> <tr> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td> Design Level 1.4.4 </td> <td> 3.1.E.12 </td> </tr> </tbody> </table>	Requirements	Tests	Design Level 1.4.4
Requirements	Tests			
Design Level 1.4.4	3.1.E.12			

Design Documentation				
Browse	Description This package provides the resources for permitting the user to move around to any particular activity in the software selection process at any time. It also provides a perspective on how the various activities fit together.			
	Design Decisions Browse must provide a graphical description of the normal sequence of activities in ASSIST processing.			
	<table border="1"> <thead> <tr> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td> Design Level 1.4.5 </td> <td> 3.1.E.11 </td> </tr> </tbody> </table>	Requirements	Tests	Design Level 1.4.5
Requirements	Tests			
Design Level 1.4.5	3.1.E.11			

Design Documentation		Description	
Backtrack ↶	This subprogram makes the previous window the current window.		
	Design Decisions Backtracking should work like popping a stack of previous windows.		
	Design Level 1.4.6	Requirements 3.4.5 ↑ ↓	Tests 2.3.1.3, 2.4.3 ↑ ↓

Design Documentation		Description	
Display Info Diagram ↶	This package provides the resources for displaying recommendations, system parameters, calculation details, and any other information of interest to the decision maker.		
	Design Decisions The information will be displayed in multiple levels of detail, permitting the decision maker to choose to see only as much information as is needed for the decision to be made.		
	Design Level 1.5	Requirements 3.1.D.3, 3.1.E.9, 3.1.E.10, 3.2.4, 3.4.2, 3.4.3, 3.4.4 ↑ ↓	Tests 2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.6, 2.4.2, 2.4.3, ↑ ↓



Design Documentation		Description	
Show Parameters ↩	This subprogram displays the system parameters and permits the user to change these parameters. New values given by the user will be validated.		
	Design Decisions This activity will be an option available to the user, but default values will be provided by ASSIST.		
	Design Level 1.5.1	Requirements 3.1.D.3, 3.1.E.9 ↑ ↓	Tests 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2, 2.4.4 ↑ ↓

Design Documentation		Description	
Show Missing Info ↩	This subprogram displays the names of features and criteria for each implementation considered for which no assessment data exists in the database.		
	Design Decisions This activity will be an option available to the user.		
	Design Level 1.5.2	Requirements 3.1.D.3, 3.1.E.9 ↑ ↓	Tests 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2, 2.4.4 ↑ ↓

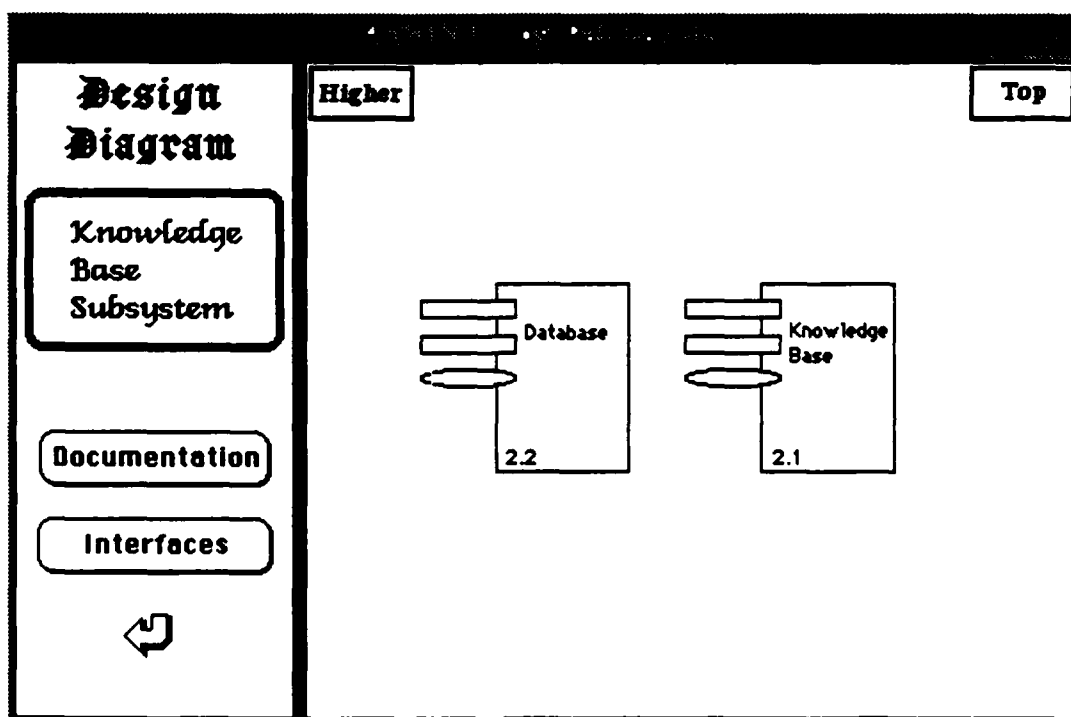
Design Documentation		Description												
Display Resources		This package contains the resources for displaying information to the user on the screen.												
		Design Decisions Headers will be provided for each different type of information.												
		Requirements <table border="1"> <thead> <tr> <th>Design Level</th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>1.5.3</td> <td>3.2.2, 3.4.2, 3.4.3, 3.4.4</td> <td>2.2.1.2, 2.3.1.6, 2.4.3</td> </tr> </tbody> </table>	Design Level	Requirements	Tests	1.5.3	3.2.2, 3.4.2, 3.4.3, 3.4.4	2.2.1.2, 2.3.1.6, 2.4.3	Tests <table border="1"> <thead> <tr> <th>Design Level</th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>1.5.3</td> <td>3.2.2, 3.4.2, 3.4.3, 3.4.4</td> <td>2.2.1.2, 2.3.1.6, 2.4.3</td> </tr> </tbody> </table>	Design Level	Requirements	Tests	1.5.3	3.2.2, 3.4.2, 3.4.3, 3.4.4
Design Level	Requirements	Tests												
1.5.3	3.2.2, 3.4.2, 3.4.3, 3.4.4	2.2.1.2, 2.3.1.6, 2.4.3												
Design Level	Requirements	Tests												
1.5.3	3.2.2, 3.4.2, 3.4.3, 3.4.4	2.2.1.2, 2.3.1.6, 2.4.3												

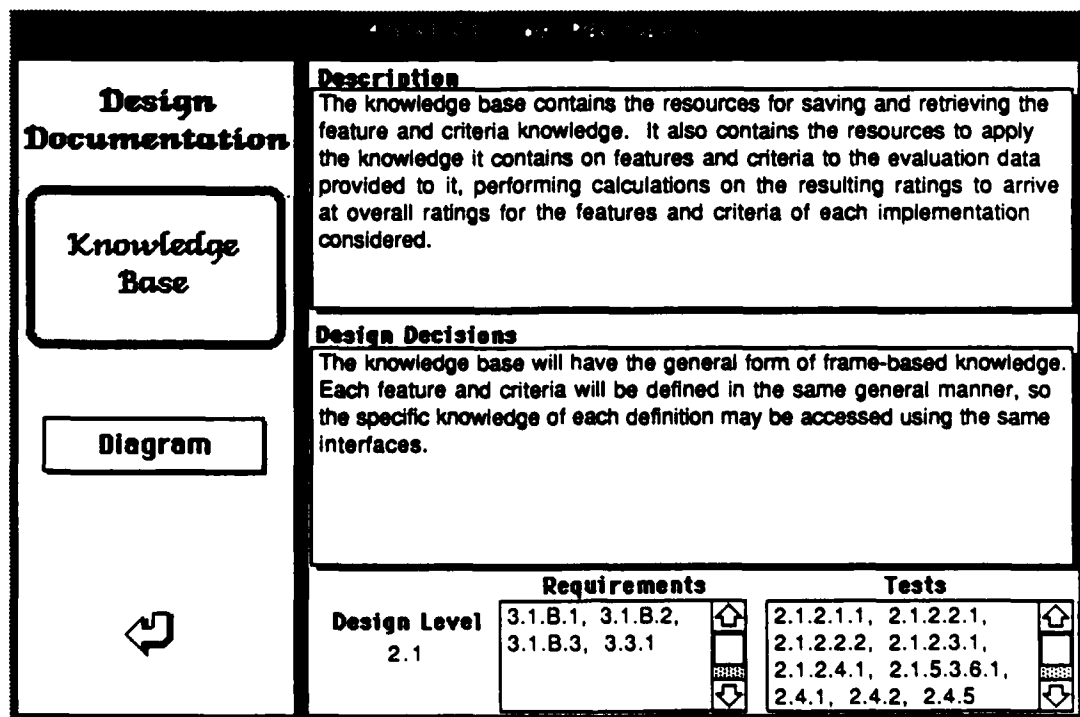
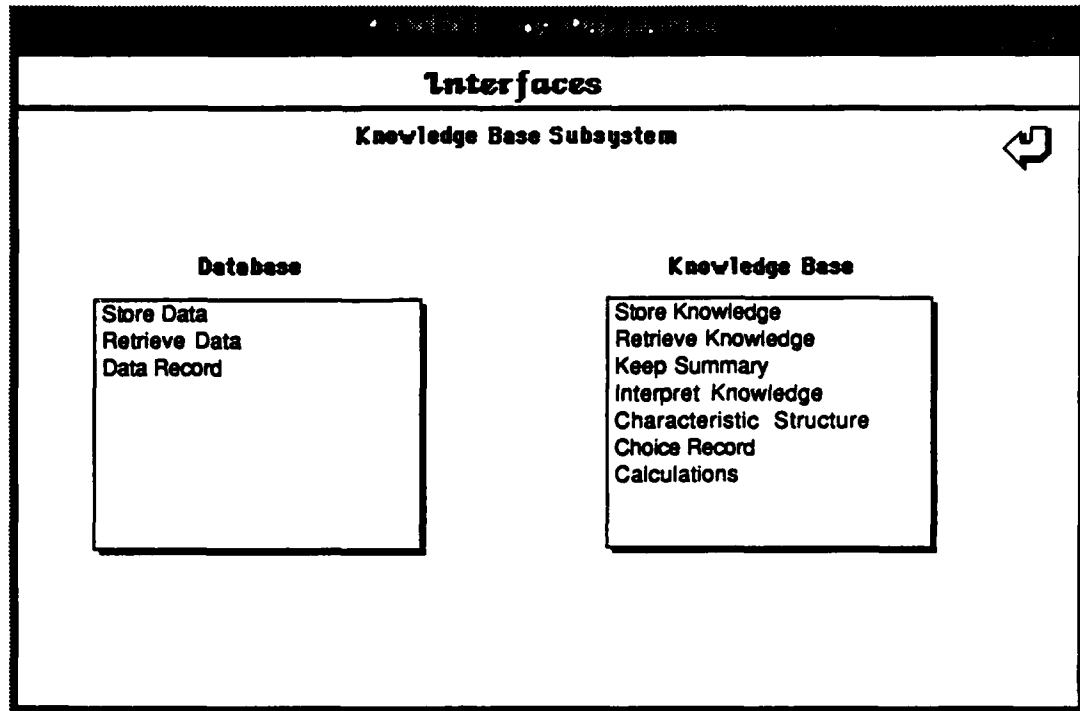
Design Documentation		Description												
Display Summary		This subprogram determines the overall rating for each implementation and prepares the summary of recommendations for display to the user. It sorts the final ratings and uses a system parameter to determine the cutoff for acceptable and unacceptable implementations. It displays results in multiple levels of detail.												
		Design Decisions All implementations in the database will be presented to the user to provide a perspective on the breadth of data in the system. Numerical ratings will not be shown to the user at the top level. When numerical ratings are shown in detail levels, they will use 2 significant digits.												
	PDL	Requirements <table border="1"> <thead> <tr> <th>Design Level</th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>1.5.4</td> <td>3.1.D.3, 3.1.E.9, 3.1.E.10, 3.1.E.13, 3.2.4</td> <td>2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.4, 2.4.2, 2.4.3</td> </tr> </tbody> </table>	Design Level	Requirements	Tests	1.5.4	3.1.D.3, 3.1.E.9, 3.1.E.10, 3.1.E.13, 3.2.4	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.4, 2.4.2, 2.4.3	Tests <table border="1"> <thead> <tr> <th>Design Level</th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>1.5.4</td> <td>3.1.D.3, 3.1.E.9, 3.1.E.10, 3.1.E.13, 3.2.4</td> <td>2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.4, 2.4.2, 2.4.3</td> </tr> </tbody> </table>	Design Level	Requirements	Tests	1.5.4	3.1.D.3, 3.1.E.9, 3.1.E.10, 3.1.E.13, 3.2.4
Design Level	Requirements	Tests												
1.5.4	3.1.D.3, 3.1.E.9, 3.1.E.10, 3.1.E.13, 3.2.4	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.4, 2.4.2, 2.4.3												
Design Level	Requirements	Tests												
1.5.4	3.1.D.3, 3.1.E.9, 3.1.E.10, 3.1.E.13, 3.2.4	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.4.2.5, 2.1.5.3.6.1, 2.2.1.3, 2.3.1.4, 2.4.2, 2.4.3												

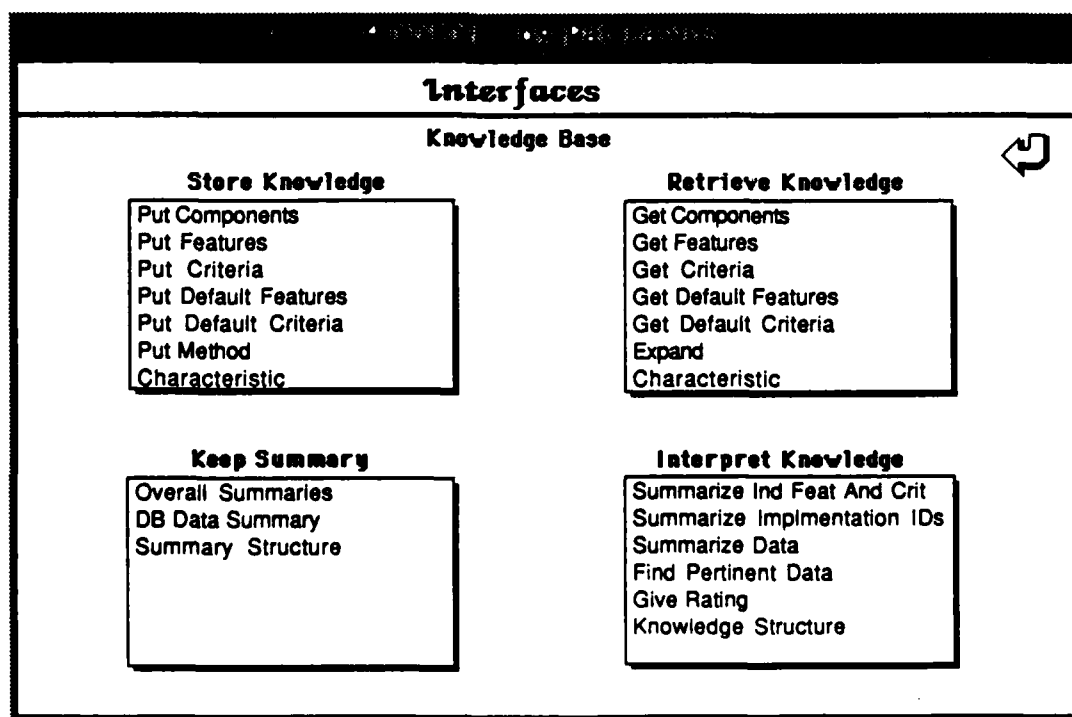
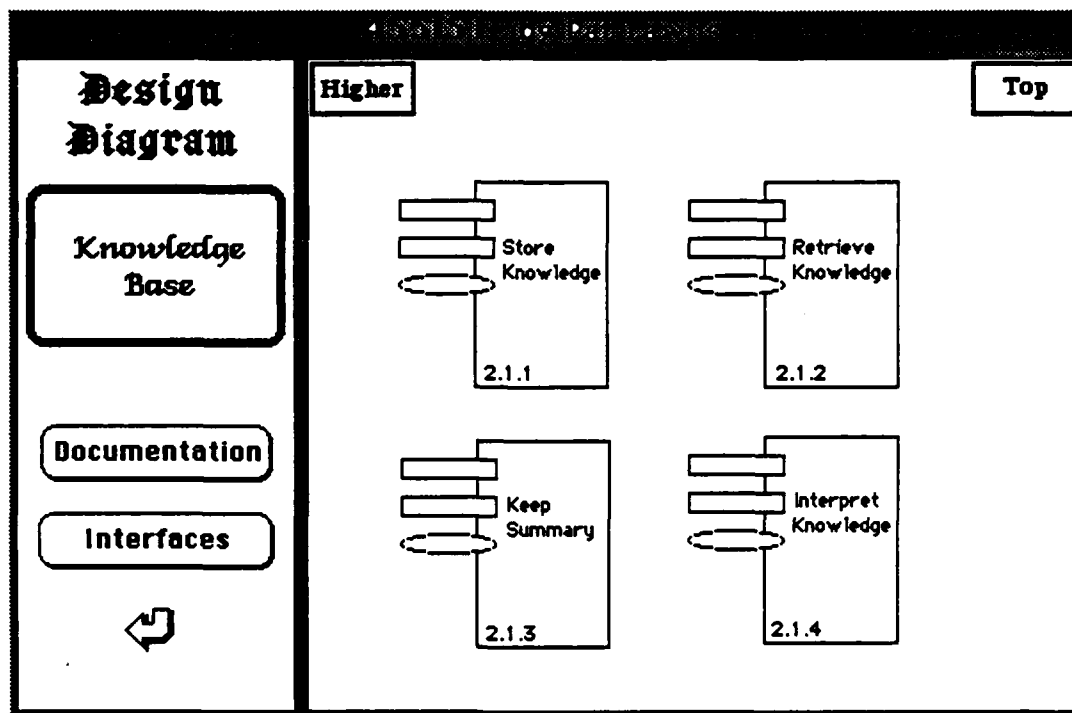
Description
At the top level, the implementations will be presented in 3 lists: the acceptable implementations (listed in order by ratings), the unacceptable implementations (listed in order by ratings), and the implementations not considered because they did not contain at least one feature specified by the user as absolutely essential. Detail levels will include explanations of rating calculations. At the lowest level of detail, the user may inspect entries in the database, and pointers will be provided to the evaluation reports which provided the data.


Program Design Language (PDL)
Subprogram Display Summary for each implementation weighted feature rating = overall feature weight * top feature ratings weighted criteria rating = overall criteria weight * top criteria ratings overall rating sum = weighted feature rating + weighted criteria rating overall weight sum = overall feature weight + overall criteria weight final rating = overall rating sum / overall weight sum end loop sort implementation IDs by ratings display acceptable implementations display unacceptable implementations display implementations not considered


Design Documentation						
Knowledge Base Subsystem Diagram ↶	Description The Knowledge Base Subsystem is composed of the database and the knowledge base. The database provides the resources for the storage and retrieval of all evaluation data. The knowledge base resources handle the storage and retrieval of all knowledge of features and criteria, as well as the interpretation of this knowledge.					
	Design Decisions The subsystem will contain the two major components: the database and the knowledge base.					
	<table border="1"> <thead> <tr> <th>Design Level</th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>2</td> <td> 3.1.A.2, 3.1.A.3, 3.1.A.4, 3.1.B.1, 3.1.B.2, 3.1.B.3, 3.2.5, 3.3.3 </td> <td> 2.1.1.2.1, 2.1.1.2.2, 2.1.1.3.1, 2.1.1.4.1, 2.1.2.1.1, 2.1.2.2.1, 2.1.2.2.2, 2.1.2.3.1 </td> </tr> </tbody> </table>	Design Level	Requirements	Tests	2	3.1.A.2, 3.1.A.3, 3.1.A.4, 3.1.B.1, 3.1.B.2, 3.1.B.3, 3.2.5, 3.3.3
Design Level	Requirements	Tests				
2	3.1.A.2, 3.1.A.3, 3.1.A.4, 3.1.B.1, 3.1.B.2, 3.1.B.3, 3.2.5, 3.3.3	2.1.1.2.1, 2.1.1.2.2, 2.1.1.3.1, 2.1.1.4.1, 2.1.2.1.1, 2.1.2.2.1, 2.1.2.2.2, 2.1.2.3.1				



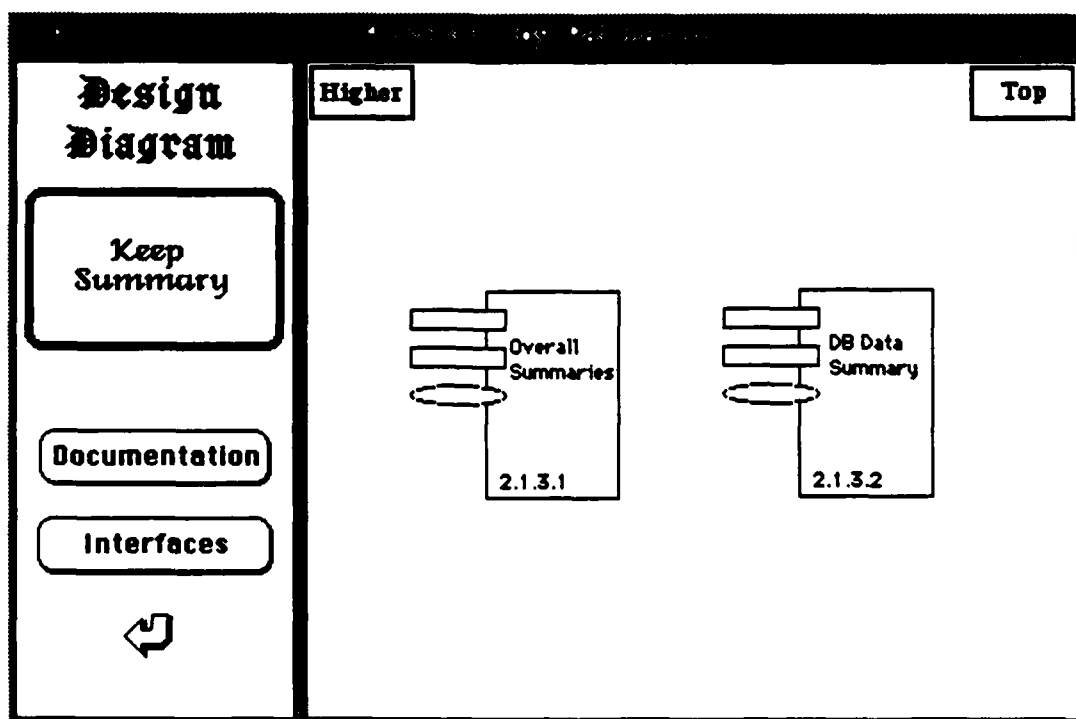




Design Documentation						
Store Knowledge	Description This package provides the resources for storing new knowledge into the knowledge base.					
	Design Decisions These resources are for the use of the system manager in setting up the system for use.					
	<table border="1"> <thead> <tr> <th></th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>Design Level 2.1.1</td> <td>3.1.B.1, 3.1.B.2</td> <td>2.1.2.1.1, 2.4.1</td> </tr> </tbody> </table>		Requirements	Tests	Design Level 2.1.1	3.1.B.1, 3.1.B.2
	Requirements	Tests				
Design Level 2.1.1	3.1.B.1, 3.1.B.2	2.1.2.1.1, 2.4.1				

Design Documentation						
Retrieve Knowledge	Description This package provides the resources for retrieving knowledge from the knowledge base when it doesn't require "massaging".					
	Design Decisions When the knowledge may be of several types, the type will be included as a part of the retrieved knowledge.					
	<table border="1"> <thead> <tr> <th></th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>Design Level 2.1.2</td> <td>3.1.B.1, 3.1.B.2</td> <td>2.1.2.1.1, 2.4.1</td> </tr> </tbody> </table>		Requirements	Tests	Design Level 2.1.2	3.1.B.1, 3.1.B.2
	Requirements	Tests				
Design Level 2.1.2	3.1.B.1, 3.1.B.2	2.1.2.1.1, 2.4.1				

Design Documentation						
Keep Summary Diagram ↶	Description This package contains the resources to be used for storing and retrieving summary information about the ratings of the various implementations as the system is processing its recommendations.					
	Design Decisions These resources are split into data summaries produced for each implementation and overall summaries showing comparisons of the various implementations.					
	<table border="1"> <thead> <tr> <th>Design Level</th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>2.1.3</td> <td>3.1.E.9</td> <td>2.1.5.3.6.1, 2.4.2</td> </tr> </tbody> </table>	Design Level	Requirements	Tests	2.1.3	3.1.E.9
Design Level	Requirements	Tests				
2.1.3	3.1.E.9	2.1.5.3.6.1, 2.4.2				

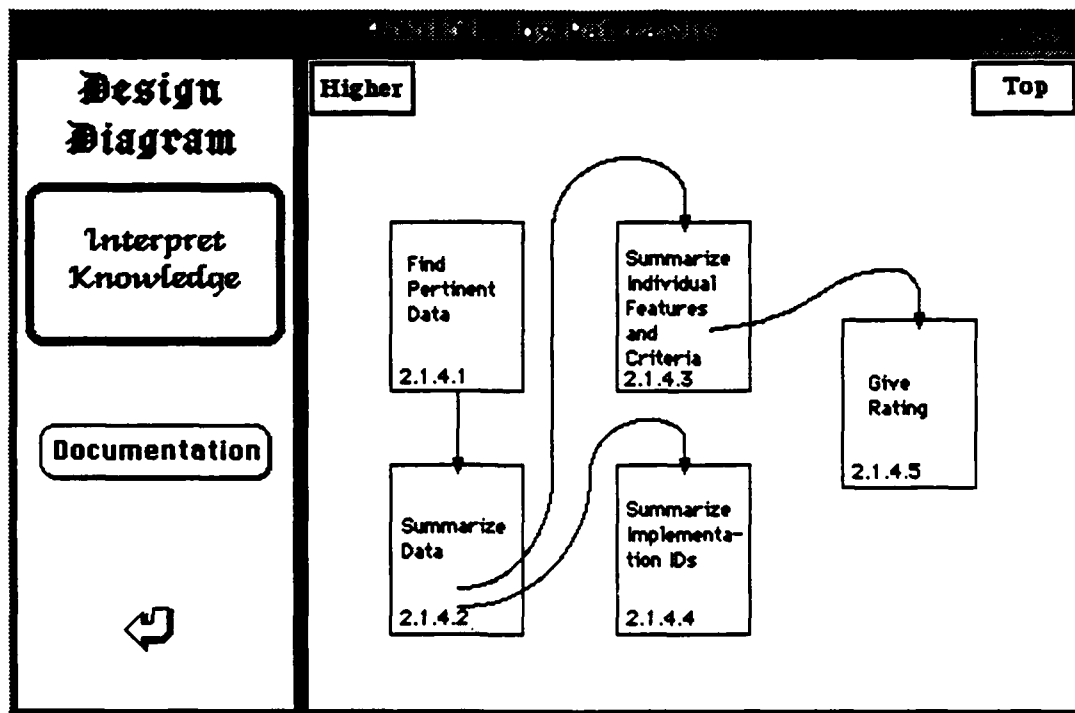


Interfaces		
Keep Summary Overall Summaries Clear Summaries Put Ratings Add to Subject Index Put Without Required Features Get Without Required Features Get Ratings Summary	D B D a t a S u m m a r y	Find Data Summary Get Desired Features Present Get Desired Features Absent Get Other Features Present Get Desired Criteria Present Get Desired Criteria Absent Get Other Criteria Present Get Criteria Rating Get Feature Rating Get Num Evals Put Feature Rating Put Criteria Rating Put Num Evals Put Desired Features Present Put Desired Features Absent Put Other Features Present Record Features Put Desired Criteria Present Put Desired Criteria Absent Put Other Criteria Present Record Criteria Summary
		

Design Documentation Overall Summaries 	Description This package provides the resources for collecting, storing, and retrieving the summary data of implementation comparisons.	
	Design Decisions The data stored and provided for each implementation must be comparable and at the same level of completeness.	
	Requirements Design Level 2.1.3.1 3.1.E.9  	Tests 2.1.5.3.6.1, 2.4.2  

Design Documentation		Description	
DB Data Summary	This package provides the resources for collecting, storing, and retrieving the summary data for individual implementations.		
	Design Decisions Absent data for chosen features and criteria will be recorded for each implementation. The data which is available but not chosen will also be recorded for each implementation.		
	Requirements Design Level 2.1.3.2	3.1.E.9	Tests 2.1.5.3.6.1, 2.4.2

Design Documentation		Description	
Interpret Knowledge Diagram	This subprogram retrieves all the pertinent data from the database for the given user choices, massages it into a usable form, performs necessary calculations, and then summarizes results.		
	Design Decisions Specific knowledge may be present for any type of data for any feature or criterion. Default knowledge may also apply to any type of data which is not handled by the specific knowledge of a particular feature or criterion.		
	Requirements Design Level 2.1.4	3.1.D.3, 3.1.E.9, 3.3.2	Tests 2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2



Design Documentation

Find Pertinent Data

PDL

Description
This subprogram retrieves all the pertinent data from the database for the given user choices.

Design Decisions
The data to be retrieved may exclude data from particular types of sources or data which was collected before a particular cut-off date. However, exclusions must apply uniformly to all data retrieved.

Design Level	Requirements	Tests
2.1.4.1	3.1.D.3, 3.1.E.9	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2

Program Design Language (PDL)

Subprogram Find Pertinent Data

```

loop for each candidate implementation
  loop for each evaluation of the implementation
    save list of desired features evaluated along with data locations
    save list of desired criteria evaluated along with data locations
  end loop
end loop

```



Design Documentation

Summarize Data

Description
This subprogram summarizes the interpreted data about all the implementations.

Design Decisions
Summaries are done separately for individual features and criteria and then the feature and criteria ratings are summarized for each implementation.

	Requirements	Tests
Design Level	3.1.D.3, 3.1.E.9, 3.3.2	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2
2.1.4.2	 	 



Design Documentation						
Summarize Individual Features and Criteria PDL ↩	Description This subprogram determines the ratings for each individual feature and criteria for a particular implementation and then summarizes the data.					
	Design Decisions Assume that each assessment for any individual feature or criterion of an implementation is of equal value, so they are averaged to arrive at a single rating for the feature or criterion.					
	<table border="1"> <thead> <tr> <th></th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>Design Level</td> <td>3.1.D.3, 3.1.E.9, 3.3.2</td> <td>2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2</td> </tr> </tbody> </table>		Requirements	Tests	Design Level	3.1.D.3, 3.1.E.9, 3.3.2
	Requirements	Tests				
Design Level	3.1.D.3, 3.1.E.9, 3.3.2	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2				

Program Design Language (PDL)
Subprogram Summarize Individual Features and Criteria <pre> loop for each feature or criterion chosen set the total ratings to 0 set the number of ratings to 0 loop for each assessment of the feature or criterion get data from database get a rating for the data add rating to the total ratings add 1 to the number of ratings end loop rating = total ratings / number of ratings save rating end loop </pre> ↩

Design Documentation									
Summarize Implementation IDs PDL ↩	Description This subprogram summarizes the interpreted data about all the implementations and arrives at a comparison of the ratings of the various implementations.								
	Design Decisions Feature and criteria calculations are done the same way: each set of detail level ratings are combined using a linear additive function to determine the rating for the top level feature or criterion. The top level ratings are then also combined using a linear additive function to determine overall feature and criteria ratings.								
	<table border="1"> <thead> <tr> <th></th> <th>Requirements</th> <th>Tests</th> </tr> </thead> <tbody> <tr> <td>Design Level</td> <td>3.1.D.3, 3.1.E.9, 3.3.2</td> <td>2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2</td> </tr> <tr> <td>2.1.4.4</td> <td> ↑ ↓ ↕ </td> <td> ↑ ↓ ↕ </td> </tr> </tbody> </table>		Requirements	Tests	Design Level	3.1.D.3, 3.1.E.9, 3.3.2	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2	2.1.4.4	↑ ↓ ↕
	Requirements	Tests							
Design Level	3.1.D.3, 3.1.E.9, 3.3.2	2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2							
2.1.4.4	↑ ↓ ↕	↑ ↓ ↕							

Program Design Language (PDL)
Subprogram Summarize Implementation IDs <pre> set the top ratings to 0 set the top weights to 0 loop for each top level feature or criterion chosen set the detail ratings to 0 set the detail weights to 0 loop for each detail feature or criterion chosen retrieve rating add weight * rating to detail ratings add weight to detail weights end loop add weight * detail ratings / detail weights to top ratings add weight to top weights end loop overall rating = top ratings / top weights </pre> ↩

Design Documentation Give Rating 	Description This subprogram produces a numerical rating based on the given data from the database, the type of the data, and the requirement specified by the user for the feature or criterion.	
	Design Decisions The algorithm used will differ among the different features and criteria and according to the type of data in the database. The knowledge base keeps the knowledge of which algorithm to use within the knowledge structure associated with each feature and criterion.	
	Requirements Design Level 2.1.4.5	Tests 2.1.4.2.2, 2.1.4.2.3, 2.1.4.2.4, 2.1.5.3.6.1, 2.4.2, 2.4.4

Design Documentation Database 	Description The database takes care of saving and retrieving all data stored in ASSIST.	
	Design Decisions Data is stored by keyword, and the keywords are the features and criteria defined in the knowledge base.	
	Requirements Design Level 2.2	Tests 2.1.1.2.1, 2.1.1.2.2, 2.1.1.3.1, 2.1.1.4.1, 2.4.1, 2.4.5

Design Documentation		Knowledge Acquisition Subsystem	
	Description The Knowledge Acquisition Subsystem provides the mechanism for acquiring all data and knowledge which will be stored in the database and knowledge base, respectively. The acquisition process will consist of getting the data, either interactively from the system manager or from a file, and converting the data into the appropriate form for storing in the system database or knowledge base.		
	Design Decisions Data is transformed into a form from which it may be stored in the database by keyword, where the keywords are the features and criteria defined in the knowledge base.		
	Design Level 3	Requirements 3.1.A.1, 3.1.A.3, 3.1.B.1, 3.1.B.2, 3.3.3, 3.4.1	Tests 2.1.1.1.1, 2.1.1.1.2, 2.1.1.2.1, 2.1.1.2.2, 2.1.1.5.1, 2.1.2.1.1, 2.1.2.2.1, 2.1.2.2.2,

Design Documentation		User Interface Subsystem	
	Description The User Interface Subsystem provides all the resources for interacting with a user. This includes window management and graphics, as well as dialog interaction.		
	Design Decisions These resources must include graphical capabilities. The use of a mouse is desirable.		
	Design Level 4	Requirements 3.1.E.2, 3.3.3, 3.4.2, 3.4.3, 3.4.4, 3.4.6	Tests 2.3.1.1, 2.3.1.4, 2.3.1.6, 2.3.3.1, 2.3.3.2, 2.4.1, 2.4.3, 2.4.5

Appendix H

Features Structure for ASSIST Prototype Version 2.0

Features

This appendix lists the features identified for and used in the ASSIST prototype. These are not expected to be exhaustive lists, but the top level list should be all-encompassing and the more detailed lists should provide a good first cut at addressing the important aspects of software features [50], [59], [61], [80], [97], [104], [161], [171].

The term *features* is used to mean characteristics of software which are used to specify **absolute** requirements for software implementations. On the other hand, the term *criteria* is used to mean characteristics of software which are used to make **relative** comparisons of similar software implementations. The criteria used in the prototype are listed in Appendix J. All features and criteria used in the prototype are defined in Appendix L.

The decision maker is initially presented with a list of 5 to 8 recommended features from the top level list. From these, those features which are most important in the selected software may be chosen. The specific features in the list presented are based on the application area which has already been chosen. Any number of the features in the initial list may be chosen, and additionally, any other features from the top level list may be added by the user, as long as the list of chosen features does not exceed 8 altogether.

For each top level feature chosen, the user will have the opportunity to specify that feature in more detail. In most cases, a more detailed list of features will be presented, and specific items may be chosen, if desired. This appendix includes an explanation of the specific handling for each of the top level features.

Top Level

analysis functions
applied standards
associated tool requirements
configuration requirements
contractual matters
cost
hardware control
management functions
numerics
options
security issues
source code sizing
timing requirements
transformation functions
user profile

analysis functions

The following list of specific analysis features (functions which may be present in the software) is presented to the user, and any number may be chosen. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

consistency checking

cross referencing

data flow analysis

mutation analysis

regression testing

requirements simulation

statistical profiling

traceability analysis

applied standards

The following list of applied standards is presented to the user, and any number may be chosen. More standards can be included in this list in the future. If no standard is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list. The determination of the rating for the applied standards feature is based on the conformance of the software to all standards chosen.

Ada (MIL-STD-1815A)

CAIS (MIL-STD-1838A)

PCIE

DIANA

GKS

PHIGS

DoD-STD-2167A

associated tool requirements

The user will be presented with the following information:

In a future version, choosing this feature will enable the specification of tools which must be available and able to work with the software to be selected. For now, choosing this feature will have no effect.

configuration requirements

The following list of specific configuration requirements is presented to the user, and any number may be chosen. For primary memory and disk capacity, the user is presented with the default value listed, and it may be changed if desired. For any other feature, the user is presented with a list of specific values for the feature which are present in the database, and from this the user must choose one. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

<u>Feature</u>	<u>Default Value</u>
host hardware	
target hardware	
host memory needed	<= 4 MB
host disk capacity needed	<= 50 MB
peripheral devices	
operating system	
support software	
distributed system	

contractual matters

The following list of specific contractual matters is presented to the user, and any number may be chosen. For the number of users, the user is presented with the default value listed, and it may be changed if desired. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

<u>Feature</u>	<u>Default Value</u>
no restrictions on users	
number of users	≥ 1
number of CPUs	≥ 1
sale of derived software	
source code available	
support available	

cost

The user will be presented with the following default value. The default may be changed, but if no action is taken, the default stands. In a future version, this may be stated as unit cost per person rather than a lump sum, if desired. It will also be possible to be specific about the component costs, including the basic software price, training costs, installation costs, and other ancillary costs associated with making the software a productive part of the user's facility.

Cost not to exceed (in U. S. dollars) 10,000.

hardware control

The user will be presented with the following information:

Choosing this feature indicates a desire that the software be able to control hardware directly. The ability to specify what may be controlled from the software will be added in a later version. For now, ASSIST will use the existence of any hardware control to arrive at a rating for this feature.

management functions

The following list of specific management features (functions which may be present in the software) is presented to the user, and any number may be chosen. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

configuration management

cost management

object management

performance monitoring

program library management

quality management

resource management

numerics

The following list of specific numerics features is presented to the user, and any number may be chosen. This list is provided for compilation systems only. For each feature chosen, if a default is given the user is presented with the default value listed, and it may be changed if desired. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

<u>Feature</u>	<u>Default Value</u>
bits in integer	≥ 16
max integer	≥ 32768
bits in float	≥ 32
bits in exponent	≥ 8
fixed point delta	≤ 0.0001
digits in float	≥ 8
long rep forms	
short rep forms	

options

The user will be presented with the following information:

In a future version, choosing this feature will enable the specification of any number of the specific options listed in the system database. For now, ASSIST will use the existence of any options to arrive at a rating for this feature.

security issues

The user will be presented with the following information:

In a future version, choosing this feature will enable the specification of security issues, such as the levels of security which must be supported by the software to be selected. For now, choosing this feature will have no effect.

source code sizing

The following list of specific source code sizing features is presented to the user, and any number may be chosen. This list is provided for compilation systems only. For each feature chosen, the user is presented with the default value listed, and it may be changed if desired. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

<u>Feature</u>	<u>Default Value</u>
lines in unit	≥ 5000
units in compile	≥ 200
entries in task	≥ 20
elements in aggregate	≥ 100
discriminants in record	≥ 10
alternatives in case	≥ 25
alternatives in select	≥ 25
instantiations of generic	≥ 10

timing requirements

The following list of timing requirements is presented to the user, and any number may be chosen. This list is provided for compilation systems only. For each feature chosen, the user is presented with the default value listed, and it may be changed if desired. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

<u>Feature</u>	<u>Default Value</u>
compiling lines of code	> 1000 lines/min
task rendezvous	< 0.00001 sec
subprogram overhead	< 0.00001 sec
exceptions overhead	< 0.00001 sec
clock resolution	< 0.000001 sec
max blocking time	< 0.0002 sec

transformation functions

The following list of specific transformation features (functions which may be present in the software) is presented to the user, and any number may be chosen. If no feature is chosen, the user will be asked if the top level feature should be unchosen. Otherwise, at least one item must be chosen from the list.

incremental compilation

editing

formatting

linking/loading

activities transformation

object transformation

program generation

user profile

The following list of user profile features is presented to the user, and any number may be chosen. If skill level is chosen, the user will be given the choice of (1) novice, (2) intermediate, or (3) expert, and one skill level must be chosen. If training is chosen, the user will be given the choice of (1) little or none, (2) moderate, or (3) extensive, and one level of training must be chosen. If no feature is chosen from the user profile list, the user will be asked if the top level feature should be unchosen. Otherwise, a choice must be made. The determination of the rating for the user profile feature is based on all choices made from the user profile list.

skill level

training

Appendix I

Criteria Structure for ASSIST Prototype Version 2.0

Criteria

This appendix lists the criteria identified for and used in the ASSIST prototype. These are not expected to be exhaustive lists, but the top level list should be all-encompassing and the more detailed lists should provide a good first cut at addressing the important aspects of assessing software criteria [14], [24], [50], [127].

The term *criteria* is used to mean characteristics of software which are used to make relative comparisons of similar software implementations. On the other hand, the term *features* is used to mean characteristics of software which are used to specify absolute requirements for software implementations. The features used in the prototype are listed in Appendix I. All features and criteria used in the prototype are defined in Appendix L.

The decision maker is initially presented with a list of 5 to 8 recommended criteria from the top level list. From these, those criteria which are most important in the selected software may be chosen. The specific criteria in the list presented are based on the application area which has already been chosen. Any number of the criteria in the initial list may be chosen, and additionally, any other criteria from the top level list may be added by the user, as long as the list of chosen criteria does not exceed 8 altogether.

For each top level criterion chosen, the user will have the opportunity to specify that criterion in more detail. A more detailed list of criteria will be presented, and the user may choose specific items in this list, if desired. If none of the detailed criteria is chosen, all information available in the database for any criterion on the detailed list will be used in the determination of a rating for the top level criterion. This appendix includes a listing of the detailed criteria for each of the top level criteria.

Top Level

correctness

efficiency

expandability

integrity

interoperability

maintainability

reliability

reusability

survivability

transportability

usability

vendor support

verifiability

correctness

The extent to which software design and implementation conform to specifications and standards.

Detailed criteria:

completeness

consistency

traceability

efficiency

The extent to which software performs its intended functions with a minimum consumption of computing resources.

Detailed criteria:

communication effectiveness

processing effectiveness

storage effectiveness

expandability

The degree to which architectural, data, or procedural design can be extended (also called extensibility).

Detailed criteria:

augmentability

generality

modularity

self documentation

simplicity

integrity

The extent to which unauthorized access to or modification of software or data can be controlled.

Detailed criteria:

security

standards compatibility

interoperability

The degree to which an APSE may provide data base objects and their relationships in forms usable by the components and user programs of another APSE without conversion.

Detailed criteria:

communication commonality

data commonality

modularity

rehostability

retargetability

maintainability

The extent to which a component facilitates updating to satisfy new requirements or to correct deficiencies.

Detailed criteria:

augmentability

communicativeness

consistency

modularity

self documentation

simplicity

structuredness

test availability

reliability

The extent to which a component can be expected to perform its intended functions in a satisfactory manner over a specified period of time.

Detailed criteria:

accuracy

completeness

consistency

fault tolerance

modularity

simplicity

reusability

The extent to which a program (or parts of a program) can be reused in other applications.

Detailed criteria:

application independence

generality

hardware independence

modularity

operating system independence

self documentation

survivability

The extent to which the software will perform and support critical functions without failures within a specified time period when a portion of the system is inoperable.

Detailed criteria:

autonomy

distributedness

fault tolerance

modularity

reconfigurability

transportability

The effort required to transfer the program from one hardware and/or software system environment to another (also called portability).

Detailed criteria:

hardware independence

modularity

operating system independence

rehostability

retargetability

self documentation

support software independence

usability

The extent to which resources required to acquire, install, learn, operate, prepare input for, and interpret output of a component are minimized.

Detailed criteria:

capacity

ease of installation

ease of use

maturity

on-line help

power

tailorability

user documentation

vendor support

The extent to which a vendor is willing and able to provide the software user with assistance to ensure that the software performs desired functions and is willing and able to support the continuing maturation of the product.

Detailed criteria:

corporate health

pricing policies

reputation

support policies

verifiability

The extent to which a component facilitates the establishment of verification criteria and supports evaluation of its performance.

Detailed criteria:

communicativeness

modularity

self documentation

simplicity

standards compatibility

structuredness

test availability

Appendix J

Glossary

Glossary

The following definitions have been adapted from several sources. In cases where these sources provided different definitions for the same term, all definitions have been included. Each definition is given in one sentence. The first definition always expresses the sense in which the term is used in ASSIST. The definitions for features (absolute characteristics) used in ASSIST are preceded by (f) and the definitions for criteria (relative characteristics) used in ASSIST are preceded by (c) [10], [14], [24], [27], [37], [45], [46], [47], [50], [57], [78], [106], [127].

accuracy - (c) A quantitative measure of the magnitude of error expressed as a function of the relative error, with a high value corresponding to a small error. The precision of computations and control. Those characteristics of software which provide the required precision in calculations and outputs.

activities transformation - (f) A software function which performs a transformation on a product of one life cycle activity to produce a product for another activity.

Ada (MIL-STD-1815A) - (f) The standard which specifies the Ada language.

alternatives in case - (f) The maximum number of individual alternatives which can be defined in a case statement.

alternatives in select - (f) The maximum number of alternatives which can be defined in a select statement.

analysis functions - (f) Software functions which provide an examination of a substantial whole to determine both qualitative and quantitative properties.

application independence - (c) The extent to which software is not dependent on the support required for a particular application. Those characteristics of software which determine

its nondependency on database system, microcode, computer architecture, and algorithms.

applied standards - (f) Standards to which software or its inputs or outputs conform.

associated tool requirements - (f) Tools which must be available and compatible with the software.

augmentability - (c) The extent to which software provides for expansion of capability for functions and data. Those characteristics of software which provide for expansion of capability for functions and data.

autonomy - (c) The extent to which software is not dependent on interfaces and functions. Those characteristics of software which determine its non-dependency on interfaces and functions.

bits in float - (f) The total number of bits used for a float representation.

bits in exponent - (f) The number of bits used for the representation of the exponent (including its sign) in a float representation.

bits in integer - (f) The number of bits used for an integer representation.

CAIS (MIL-STD-1838A) - (f) The standard which specifies the Common APSE Interface Set, a set of interfaces to the APSE kernel.

capacity - (c) The extent of the upper and lower limits of the functions implemented by a tool.

clock resolution - (f) The amount of time distinguishing (the difference between) two consecutive clock times.

communication commonality - (c) The degree to which standard interfaces, protocols, and bandwidths are used. Those

characteristics of software which provide for the use of interface standards for protocols, routines, and data representations.

communication effectiveness - (c) The extent to which software performs its intended functions with a minimum consumption of communications resources. Those characteristics of the software which provide for minimum utilization of communications resources in performing functions.

communicativeness - (c) The degree to which the program provides feedback while it is operating to keep the user informed of the functions being performed.

compiling lines of code - (f) The number of lines of source code which are compiled in a minute (wall clock time).

completeness - (c) The extent to which a component provides the complete set of operations necessary to perform a function. The degree to which full implementation of required function has been achieved. Those characteristics of software which provide full implementation of the functions required.

configuration management - (f) A software function which establishes baselines for configuration items, controls the changes to these baselines, and controls releases to the operational environment.

configuration requirements - (f) Those specific components of system hardware and/or software which are required in order for the software to function correctly.

consistency - (c) The extent to which uniform design and documentation techniques have been used throughout the software development project. The use of uniform design and documentation techniques throughout the software development project. Those characteristics of software which provide for uniform design and implementation techniques and notation.

- consistency checking** - (f) A software function which determines whether or not an entity is internally consistent in the sense that it is consistent with its specification.
- contractual matters** - (f) Features determining the legal use of and support provided for software which may be specified in a contract with the vendor at the time of purchase.
- corporate health** - (c) The extent to which it is reasonable to assume that the vendor will remain in business with the ability to continue the current level of customer support.
- correctness** - (c) The extent to which software design and implementation conform to specifications and standards. The extent to which a program satisfies its specification and fulfills the customer's mission objectives. The extent to which software is free from design defects and from coding defects; that is, fault free. Agreement between a component's total response and the stated response in the functional specification (functional correctness), and/or between the component as coded and the programming specification (algorithmic correctness).
- cost** - (f) The total price associated with the purchase and productive use of the software (including the basic software price, training costs, installation costs, and any other ancillary costs associated with making the software a productive part of the user's facility).
- cost management** - (f) A software function which manages cost functions (such as the cost organization structure and the cost estimation methodology).
- criteria** - Characteristics of software which are used to make relative comparisons of similar software implementations.
- cross referencing** - (f) A software function which references entities to other entities by logical means.
- data commonality** - (c) The extent to which standard data structures and types are used throughout the program. The

use of standard data structures and types throughout the program. Those characteristics of software which provide for the use of interface standards for data representations.

data flow analysis - (f) A software function which analyzes the formal requirements statements to determine interface consistency and data availability.

DIANA - (f) The standard which specifies a Descriptive Intermediate Attributed Notation for Ada, an abstract data type such that each object of the type is a representation of an intermediate form of an Ada program.

digits in float - (f) The largest number of decimal digits which may be represented by a float.

discriminants in record - (f) The maximum number of discriminants which can be defined for a single record type.

distributed system - (f) A system in which software functions are geographically or logically separated within the system.

distributedness - (c) The degree to which software functions are geographically or logically separated within the system. Those characteristics of software which determine the degree to which software functions are geographically or logically separated within the system.

DoD-STD-2167A - (f) The standard which establishes uniform requirements for software development that are applicable throughout the system life cycle.

ease of installation - (c) The relative ease with which a software product may be integrated into its operational environment and tested in this environment to ensure that it performs as required.

ease of use - (c) The relative ease with which a novice user can become an effective user of the program.

editing - (f) A software function which provides for selective revision of computer-resident data (the data may be textual, graphical, some internal representation, etc.).

efficiency - (c) The extent to which software performs its intended functions with a minimum consumption of computing resources. The amount of computing resources and code required by a program to perform its function. The relative extent to which a resource is utilized. The ratio of actual utilization of the system resources to optimum utilization.

elements in aggregate - (f) The maximum number of elements which can constitute an aggregate.

entries in task - (f) The maximum number of entries which can be defined in a single task.

exactness - The measure of assuredness that a component does no more than it was specified to do and does not contain malicious code.

exceptions overhead - (f) The execution overhead time which is attributable to the presence of exception handlers in the unit.

expandability (extensibility) - (c) The degree to which architectural, data, or procedural design can be extended. The relative effort to increase the software capability or performance by enhancing current functions or by adding new functions or data. The extent to which a component allows new capabilities to be added and existing capabilities to be easily tailored to user needs.

fault tolerance - (c) The extent to which the system has the built-in capability to provide continued correct execution in the presence of a limited number of hardware or software faults. Those characteristics of software which provide for continuity of operations under and recovery from non-nominal conditions. The protection of a component from itself, user errors, and system errors. The ability to recover and provide meaningful diagnostics in the event of unforeseen situations. The damage that occurs when the program encounters an error.

features - Characteristics of software which are used to specify absolute requirements for software implementations.

fixed point delta - (f) The smallest interval which may be used to distinguish among fixed point values.

formatting - (f) A software function which arranges data according to predefined and/or user-defined conventions.

generality - (c) The breadth of the potential application of program components. Those characteristics of software which provide breadth to the functions performed with respect to the application.

GKS - (f) The standard which specifies the Graphical Kernel System, a graphics system which allows programs to support a wide variety of graphics devices and which is defined independently of programming languages.

hardware control - (f) The ability of the software to control hardware directly (such as interrupts, bit manipulations, file servers, task scheduling, preemption, etc.).

hardware independence - (c) The degree to which the software is decoupled from the hardware on which it operates. Those characteristics of software which determine its nondependency on specific hardware. The degree to which hardware dependencies are isolated in a distinct library unit.

host disk capacity needed - (f) The combined storage size (in megabytes) required of the on-line disk units of the host hardware to ensure that the software will run properly.

host hardware - (f) The specification of the manufacturer and model of the computer hardware which will serve as the development platform for the software to be developed.

host memory needed - (f) The size (in megabytes) required of the primary memory of the host hardware to ensure that the software will run properly.

incremental compilation - (f) A software function which produces new object code for a particular source code unit from the previous object code for that unit and a set of specified changes to the source code which produced the original object code.

instantiations of generic - (f) The maximum number of times a single generic unit can be instantiated.

integrity - (c) The extent to which unauthorized access to or modification of software or data can be controlled. The extent to which the software will perform without failures due to unauthorized access to the code or data within a specified time period. The probability that the system will perform without failure and will protect the system and data from unauthorized access.

interoperability - (c) The degree to which an APSE may provide data base objects and their relationships in forms usable by the components and user programs of another APSE without conversion. The extent to which two or more systems have the ability to exchange information and to mutually use the information that has been exchanged. The effort required to couple one system to another. The relative effort to couple the software of one system to the software of another system. The probability that two or more systems can exchange information under stated conditions and use the information that has been exchanged.

lines in unit - (f) The maximum number of source code lines which can be compiled in one compilation unit.

linking/loading - (f) A software function which creates a load/executable module on the host machine from one or more independently translated object modules or load modules by resolving cross-references among the object modules, and possibly relocating elements.

long rep forms - (f) The ability to specify a number (integer or float) which will be represented using more total bits than is

used by numbers of the same base type without the "long" designation.

maintainability - (c) The extent to which a component facilitates updating to satisfy new requirements or to correct deficiencies. The effort required to locate and fix an error in a program. The ease of effort for locating and fixing a software failure within a specified time period. The ease with which software can be maintained. The probability that the system can be restored to a specified condition within a specified amount of time.

malicious code - operations which covertly damage or attempt to by-pass system security.

management functions - (f) Software functions which aid the management or control of system/software development.

maturity - (c) The extent to which a component has been used in the development of deliverable software by typical users and to which the feedback from that use has been reflected in modifications to the component.

max blocking time - (f) The maximum amount of overhead time used by the runtime system to block a task.

max integer - (f) The maximum number which may be represented as an integer.

modularity - (c) The extent to which software is composed of discrete components such that a change to one component has minimal impact on other components. The extent to which a component is implemented in a hierarchical structure in which identifiable functions are isolated in separate compilation units. The functional independence of program components. Those characteristics of software which provide a structure of highly cohesive components with optimum coupling.

mutation analysis - (f) A software function which applies test data to a program and its "mutants" (programs that contain one or more likely errors) in order to determine test data adequacy.

- no restrictions on users** - (f) Not disallowing or constraining the use of the software by a particular class of users (such as people not employed by the purchasing organization).
- number of CPUs** - (f) The total number of computers which may legally serve as the residence for a particular software component.
- number of users** - (f) The maximum number of users permitted simultaneous execution of a single purchased copy of the software.
- numerics** - (f) Software features which determine the computational capabilities of the software.
- object management** - (f) A software function which manages a collection of interrelated data (objects) stored together with controlled redundancy, serving one or more applications and independent of the programs using the data (objects).
- object transformation** - (f) A software function which performs a transformation on a particular system object to produce another system object.
- on-line help** - (c) The extent to which user documentation is readily available to the user from the program while it is operating.
- operating system** - (f) The specification of the name and version of the operating system under which the software will run.
- operating system independence** - (c) The degree to which the program is independent of operating system characteristics. Those characteristics of software which determine its nondependency on a specific operating system. The degree to which operating system dependencies are isolated in a distinct library unit.
- options** - (f) Software features whose specified values (each of which causes the software to execute in a somewhat different, yet controlled, manner) are set by the user.

PCTE - (f) The standard which specifies the Portable Common Tool Environment, a hosting structure defined by a set of program-callable primitives which support the execution of programs in terms of a virtual, machine independent level of comprehensive facilities.

performance monitoring - (f) A software function which monitors the performance characteristics of the finished product.

peripheral devices - (f) The hardware devices which are attached to and work with the computer but are not an integral part of it (such as printers, terminals, etc.).

PHIGS - (f) The standard which specifies the Programmers Hierarchical Interactive Graphics Standard, a sophisticated graphics support system that controls the definition, modification, and display of hierarchical graphics data.

power - (c) The extent to which a component has capabilities, such as default options and wild card operations, that contribute to the effectiveness of the user.

pricing policies - (c) The degree to which the vendor's prices for product support and upgrades are reasonable and in accordance with accepted practice within the software industry.

processing (execution) effectiveness - (c) The extent to which software performs its intended functions with a minimum consumption of processing resources. The run-time performance of a program. Those characteristics of the software which provide for minimum utilization of processing resources in performing functions. The choice between alternative algorithms based on those taking the least amount of time.

program generation - (f) A software function which provides the translation or interpretation used to construct computer programs (such as language translator generator, syntax

analyzer generator, code generator generator, environment definition generator, user interface generator, etc.).

program library management - (f) A software function which performs the creation, manipulation, display, and deletion of the various components of a program library.

quality management - (f) A software function which manages the determination of the achieved level of quality in deployed software systems.

reconfigurability - (c) The extent to which software provides for continuity of system operation when one or more processor, storage units, or communication links fails. Those characteristics of software which provide for continuity of system operation when one or more processors, storage units, or communication links fails.

regression testing - (f) A software function which performs the rerunning of tests in order to detect errors spawned by changes or corrections made during software development and maintenance.

rehostability - (c) The extent to which an APSE component may be installed on a different host or different operating system with a minimum of reprogramming. The ability of an APSE component to be installed on a different host or different operating system with needed reprogramming localized to the KAPSE or machine dependencies.

reliability - (c) The extent to which a component can be expected to perform its intended functions in a satisfactory manner over a specified period of time. The extent to which a program can be expected to perform its intended function with required precision. The extent to which the software will perform without any failures within a specified time period. The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability that the system will perform as intended under stated conditions for a specified period of time.

reputation - (c) The degree of confidence expressed by program users in the vendor's willingness and ability to provide support for the program.

requirements simulation - (f) A software function which executes code-enhanced requirements statements to examine functional interfaces and performance.

resource management - (f) A software function which manages the resources attributed to an entity.

retargetability - (c) The extent to which an APSE component may accomplish its function with respect to another target with a minimum of modification. The ability of an APSE component to accomplish its function with respect to another target.

reusability - (c) The extent to which a program (or parts of a program) can be reused in other applications. The relative effort to convert a software component for use in another application. The relative effort to adapt software for use in another application.

sale of derived software - (f) Disallowing or constraining the conditions under which some portion of the purchased software may be included in software provided by the purchaser to a third party.

security - (c) The extent of protection of computer hardware and software from accidental or malicious access, use, modification, destruction, or disclosure. The availability of mechanisms that control or protect programs and data.

security issues - (f) Features which affect the use of the software in a classified environment.

self documentation - (c) The degree to which the source code provides meaningful documentation. Those characteristics of software which provide explanation of the implementation of functions. The technical data, including on-line, documentation, listings, and printouts, which serve the purpose of elaborating the design or details of a component.

short rep forms - (f) The ability to specify a number (integer or float) which will be represented using fewer total bits than is used by numbers of the same base type without the "short" designation.

simplicity - (c) The extent to which the complexity of a system or system component (determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, the type of data structures, and other system characteristics) is kept to a minimum. The degree to which a program can be understood without difficulty. Those characteristics of software which provide for definition and implementation of functions in the most noncomplex and understandable manner.

skill level - (f) The level of experience in using similar software.

source code available - (f) The possibility that the source code of the software can be purchase.

source code sizing - (f) The limits imposed on the size of selected components of the software.

standards compatibility - (c) The degree to which the program conforms to specific standards.

statistical profiling - (f) A software function which provides the analysis of a program to determine statement types, number of occurrences of each statement type, and the percentage of each statement type in relation to the complete program.

storage effectiveness - (c) The extent to which software performs its intended functions with a minimum consumption of storage resources. Those characteristics of the software which provide for minimum utilization of storage resources. The choice between alternative source code constructions based on those taking the minimum number of words of object code or in which the information-packing is high.

structuredness - (c) The degree to which the program is constructed of a basic set of control structures, each one having one entry point and one exit.

subprogram overhead - (f) The overhead time involved in executing a subprogram call.

support available - (f) The possibility of purchasing support for the software from the vendor on a continuing basis.

support policies - (c) The type and extent of support provided by the vendor for the software.

support software - (f) The specification of the name and version of the support software required to work with the software in question to ensure proper functionality.

support software independence - (c) The degree to which the program is independent of nonstandard programming language features and other environmental constraints. Those characteristics of software which determine its nondependency on specific support software in the environment (utilities, input and output routines, libraries). The degree to which support software dependencies are isolated in a distinct library unit.

survivability - (c) The extent to which the software will perform and support critical functions without failures within a specified time period when a portion of the system is inoperable. The extent to which software will continue performing when a portion of the system has failed.

tailorability - (c) The extent to which the user interface of the program may be altered to conform to the preferences of the user.

target hardware - (f) The specification of the manufacturer and model of the computer hardware on which the software to be developed will be executed.

task rendezvous - (f) The overhead time required to accomplish a task rendezvous.

test availability - (c) The extent to which tests are available to verify that a program functions in accordance with its requirements. The extent to which tests are available to support the evaluation of a program's performance with respect to specific verification criteria.

timing requirements - (f) The limits imposed on the execution time of selected components of the software.

traceability - (c) The ability to trace a design representation or actual program component back to requirements. Those characteristics of software which provide a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment.

traceability analysis - (f) A software function which checks for internal consistency within the software requirements specification.

training - (f) The amount of training required to be able to use the software productively.

transformation functions - (f) Software functions which describe how the subject is manipulated to accommodate the user's needs.

transportability (portability) - (c) The effort required to transfer the program from one hardware and/or software system environment to another. The relative effort to transport the software for use in another environment. The extent to which a component can be adapted for use in another environment. The extent to which a component may be installed on a different APSE without change in functionality.

units in compile - (f) The largest number of compilation units which can be involved in a single compile.

usability - (c) The extent to which resources required to acquire, install, learn, operate, prepare input for, and interpret output

of a component are minimized. The effort required to learn, operate, prepare input, and interpret the output of a program. The relative effort for using software (training and operation). The probability that users can operate the system under specified conditions without user error given they have received specified training.

user documentation - (c) The extent to which documentation conveys to the end user of a system instructions for using the system to obtain desired results. The technical data which serve the purpose of elaborating the design or details of a component to the user.

user profile - (f) Characteristics required of the user in order to use the software productively.

vendor support - (c) The extent to which a vendor is willing and able to provide the software user with assistance to ensure that the software performs desired functions and is willing and able to support the continuing maturation of the product.

verifiability - (c) The extent to which a component facilitates the establishment of verification criteria and supports evaluation of its performance. The effort required to test a program to ensure that it performs its intended function. The relative effort to verify the specified software operation and performance. The extent to which the specified system operation and performance determine the conditions and criteria for tests. The extent to which a component facilitates the evaluation of its correctness, completeness, and exactness.

Appendix K

Suggested Features and Criteria by Application

Suggested Features and Criteria

This appendix includes lists of the suggested features and criteria, along with their respective default weights, which are initially presented to a user. The particular pair of lists presented depend on the application area chosen. The suggestions given are an initial estimate of the importance usually placed on particular features and criteria for the given application area. These may be modified as suggested by experience with ASSIST.

The term *features* is used to mean characteristics of software which are used to specify **absolute** requirements for software implementations. The term *criteria* is used to mean characteristics of software which are used to make **relative** comparisons of similar software implementations. The features used in the prototype are listed in Appendix I, and the criteria used are listed in Appendix J. All the features and criteria are defined in Appendix L.

The weights for both features and criteria range from 1 to 10, where a weight of 1 means the feature or criterion is marginally important, a weight of 5 means it is moderately important, and a weight of 10 means it is critically important. When a feature is weighted with a 10, it means it is positively required, and no software should even be considered which does not contain that feature.

Hard Real Time

<u>Feature</u>	<u>Default Weight</u>
analysis functions	5
applied standards	5
hardware control	8
source code sizing	5
timing requirements	10
user profile	5

<u>Criterion</u>	<u>Default Weight</u>
correctness	5
efficiency	10
integrity	8
maintainability	5
reliability	8
usability	5
vendor support	8
verifiability	10

Soft Real Time

<u>Feature</u>	<u>Default Weight</u>
analysis functions	5
cost	5
source code sizing	5
timing requirements	7
user profile	5

<u>Criterion</u>	<u>Default Weight</u>
correctness	5
efficiency	8
integrity	8
maintainability	5
reliability	5
usability	5
vendor support	8
verifiability	5

Math Intensive

<u>Feature</u>	<u>Default Weight</u>
analysis functions	8
applied standards	5
cost	5
numerics	10
source code sizing	7
user profile	5

<u>Criterion</u>	<u>Default Weight</u>
correctness	8
integrity	5
maintainability	5
transportability	5
usability	5
vendor support	8

Information Intensive

<u>Feature</u>	<u>Default Weight</u>
contractual matters	5
cost	5
management functions	8
source code sizing	5
user profile	5

<u>Criterion</u>	<u>Default Weight</u>
correctness	5
integrity	10
maintainability	5
transportability	5
usability	8
vendor support	8

General

<u>Feature</u>	<u>Default Weight</u>
analysis functions	5
applied standards	5
cost	5
management functions	5
source code sizing	5
user profile	5

<u>Criterion</u>	<u>Default Weight</u>
correctness	5
expandability	5
integrity	5
maintainability	5
transportability	5
usability	5
vendor support	5

Appendix L

User Manual for ASSIST Prototype Version 2.0

Table of Contents

Foreword	2
Introduction	
Starting and Quitting	3
System Initialization	4
Buttons and System Functions	5
Specifying Selection Parameters	
Context Choices	12
Application Area	15
Choose Features	16
Weight Features	18
Choose Criteria	20
Weight Criteria	22
Features in Database	24
Criteria in Database	25
Definitions	26
Specifying Numerical Values	27
Displaying Results	28
Glossary	29

Foreword

The Ada Software Selection assIStAnt (ASSIST) is a type of program often referred to as a decision support system (DSS) because its main purpose is to provide support to someone who is about to make an important decision. It is similar to the type of program called an expert system, but, unlike an expert system, ASSIST will not necessarily recommend just one solution to the problem at hand. It will use information you provide to determine the important characteristics of the Ada support software to be selected and how to weight the importance of each characteristic. You will specify the type of software to be selected. This could be anything from a single software tool to a complete software support environment. Then you specify the important features and criteria to be used in the software evaluation, as well as a weight for each. ASSIST will then search its database of information, derived from evaluations performed on numerous software products, and it will determine those products of the specified type which provide an acceptable level of the characteristics you deemed important. The program will provide a list of all software products which have the required characteristics, based on the information in the database. It will also indicate which other products of the specified type, which are also included in the database, were rejected as unacceptable and why.

This manual is organized by the windows you see as you use ASSIST. The window in question is shown at the top of a page. This is followed by an explanation of the purpose of the window and what you may do while it is displayed. Other items of interest are also discussed.



Warning

If you are an experienced HyperCard user, you should know that the usual HyperCard commands will work with ASSIST. However, be aware that you could cause ASSIST severe problems by executing a command which is not available to the regular ASSIST user. Every command that you need is available as a part of ASSIST. If you use HyperCard commands, it is your responsibility to be sure you do not destroy system integrity.

Starting

ASSIST uses the application HyperCard (not included), and HyperCard runs on a Macintosh Plus, a Macintosh SE, or any of the Macintosh II models.

The first time you use ASSIST you should copy it from the original disk to a hard disk. If you do not know how to perform this operation, consult your Macintosh owner's guide or else get help from an experienced Macintosh user.

Once ASSIST is loaded on your hard disk, you should see two icons (pictures) in your folder which look like those at the right.



ASSIST



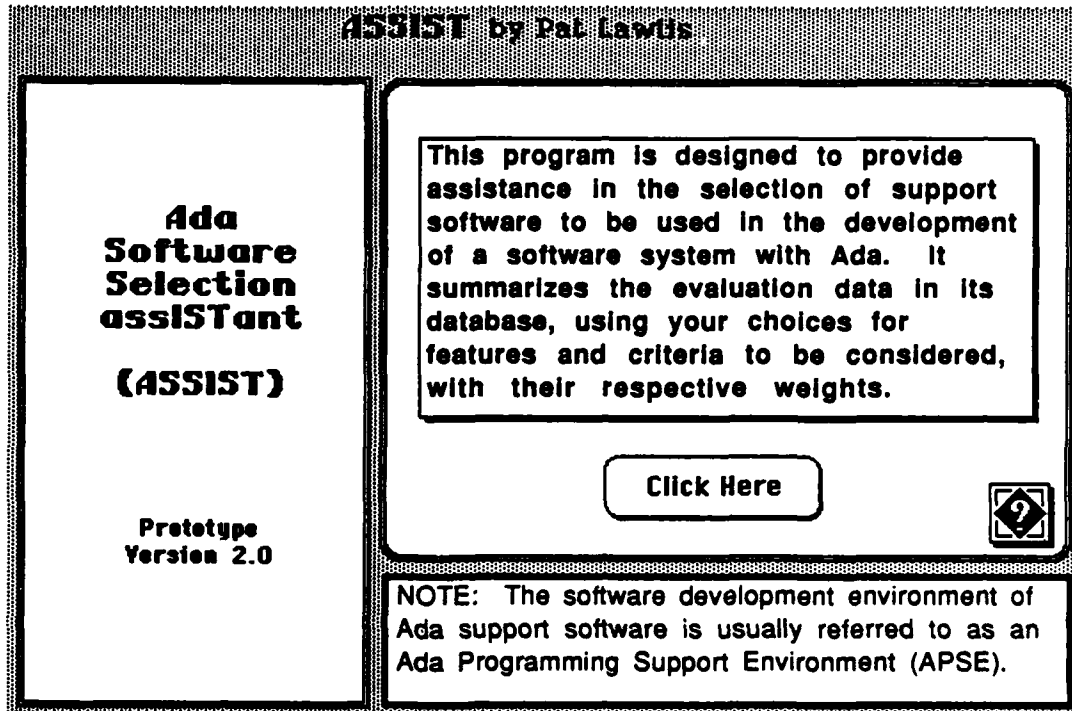
Save Stack

To start ASSIST simply double click on the icon labeled "ASSIST". The "Save Stack" is used if you save data from ASSIST, but you do not need to do anything with it. ASSIST will not be able to perform the save operation unless a stack named "Save Stack" is in the same folder with it.

Quitting

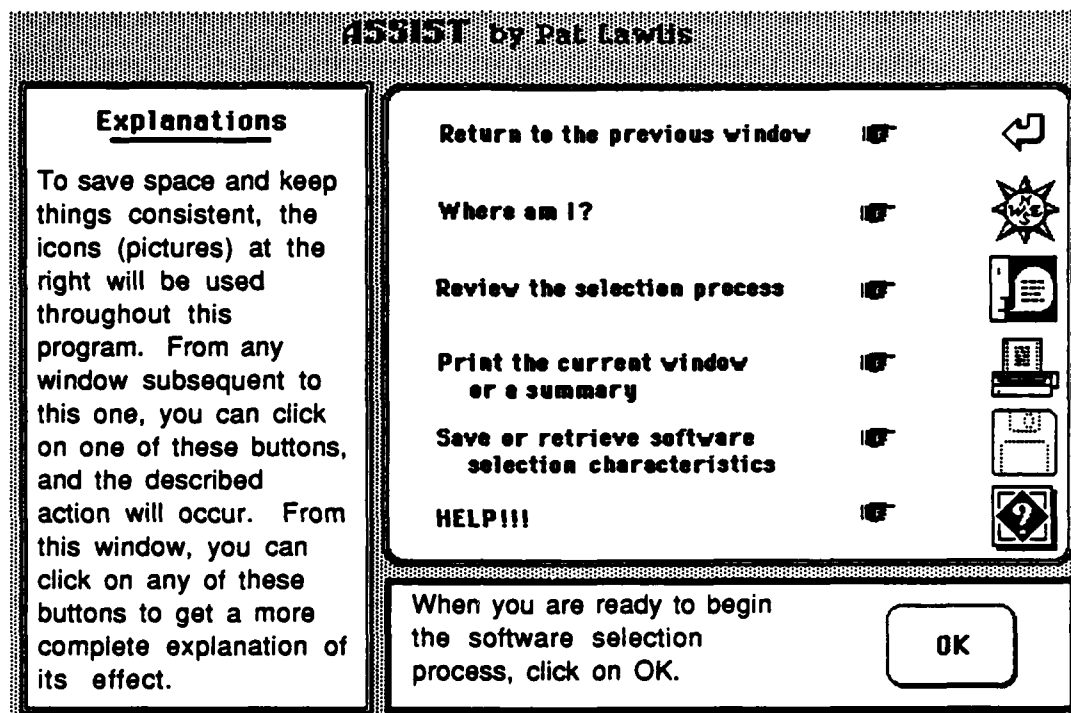
You may quit from ASSIST at any time by clicking on the Help icon (shown at the right) and then clicking on the button which says "Quit ASSIST".





When ASSIST is first started, it still contains data from the last time it was used. This permits a system analyst to examine the stack (program) after any glitches or anomalies should occur.

To prepare the stack for your new session, click on "Click Here". After a short pause, the system will be ready for you.



An important objective in presenting recommendations to a decision maker is to provide information in small enough "chunks" to be easy to understand, but at the same time to make the information useful by providing ease and flexibility in getting to the next relevant chunk. ASSIST meets this objective by providing one window at a time, where each window presents only one type of coordinated information, but by making it easy to get to any related window. The ease of transition from one window to the next is provided by the user clicking the mouse on a "button". A button can be marked with either a little picture, called an icon, or with words describing the action taken when the button is pressed (or clicked). The 6 icons described in the window shown above are present in almost every window, so any of them can be activated at any time while using ASSIST. They are described in more detail in the following pages.

The "Return" icon is pictured at the right.

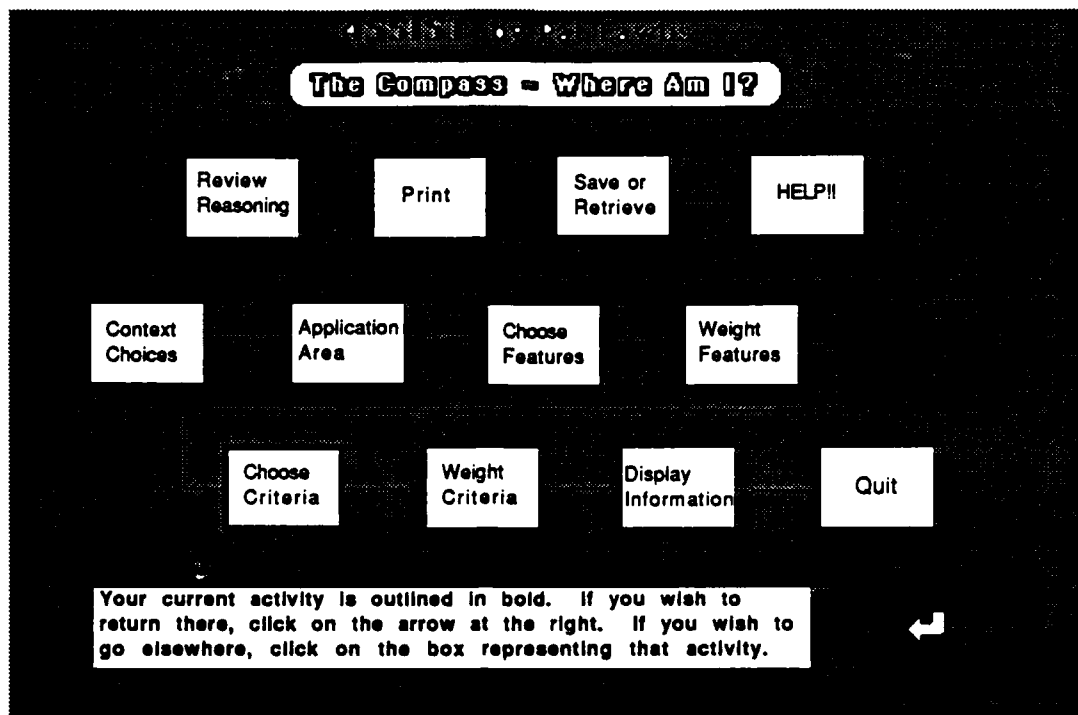


Clicking on the Return icon permits you to backtrack through the windows preceding the current window.

This may not work exactly as you would expect if you use it after retrieving specifications which were previously saved. This is because the previous windows have been replaced with new windows. If Return tries to backtrack to a previous window and it is not found, nothing will happen.

The Return button will be disabled while you are in the middle of specifying detail features, criteria, and weights. If detail processing could be interrupted or terminated, system integrity would be in jeopardy.

If you want to go back to a window you looked at BEFORE the last one, the Compass icon may be faster. You can click on the Compass graphic to move to any set of windows.



The "Compass" icon is pictured at the right.



Clicking on the Compass will take you to the "where am I" compass window, shown above. This will give you a graphic of where you have been and where you can go in the program (in the above graphic "Choose Criteria" is where you have been). You can move to any of the activities shown in the graphic just by clicking on it.

ASSIST by Pat Lawlis

..... Type of software to be selected Application area chosen Features chosen and respective weights Criteria chosen and respective weights Parameters used by the system Lowest acceptable rating is 0.9 Weight for the combined feature rating is 1 Weight for the combined criteria rating is 1	Review Scroll by clicking on the arrows or moving the box along the scroll bar. When you have finished reviewing, click on the return arrow below.  

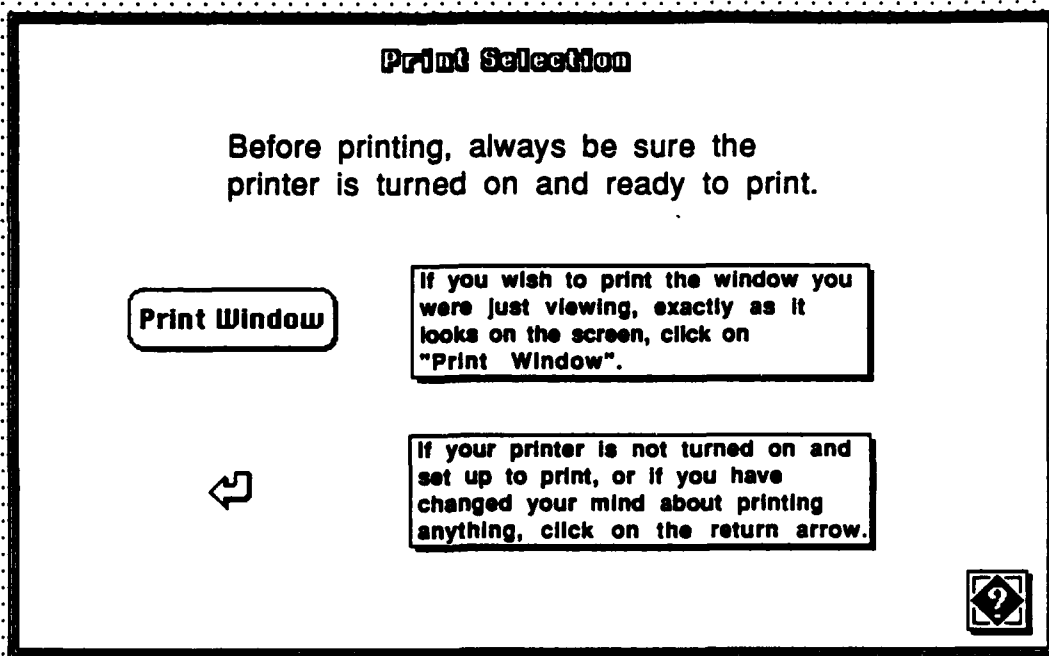
The "Review" icon is pictured at the right.



Clicking on the Review icon will show all the choices you have made so far. This includes features, criteria, and weights chosen, as well as the system parameters used to arrive at the recommendations.

The above graphic shows the Review window before any choices have been made. The system parameters given are the defaults.

The Review window does NOT show a graphic of the selection process. For that type of information, use the Compass icon. The recommendations made by ASSIST will also not be shown here. For this information, click on the Compass icon and then choose "Display Information".

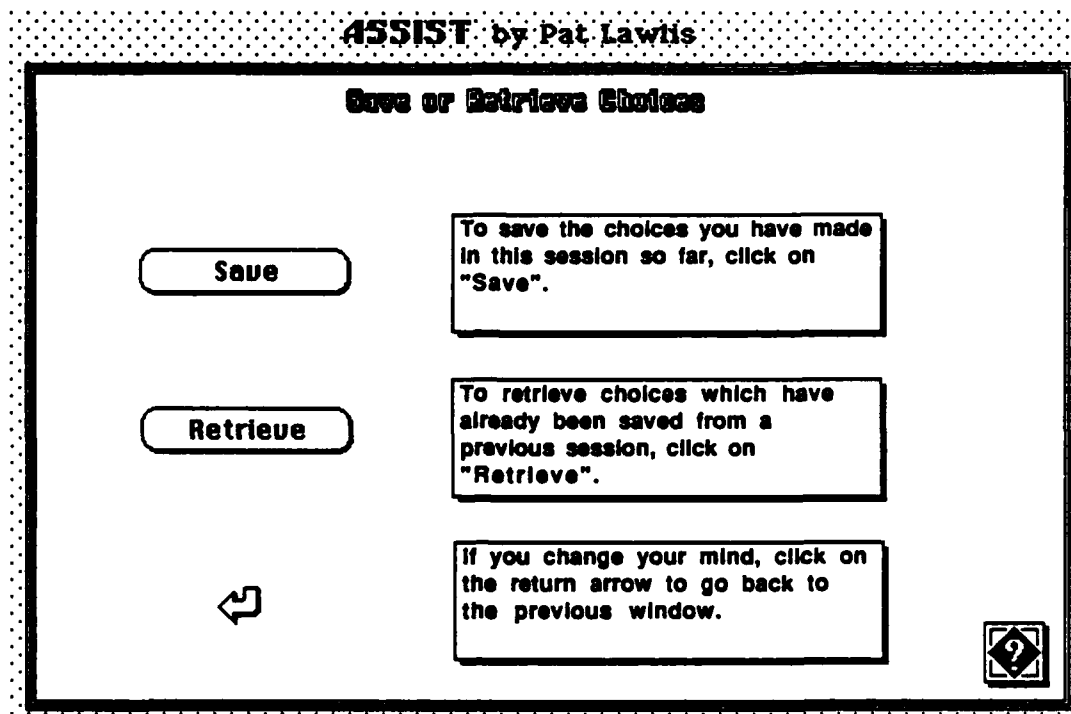
ASSIST by Pat Lawlis

The "Printer" icon is pictured at the right.



Clicking on the Printer icon will get you to the window shown above. It will currently permit you to print only the contents of the window you were viewing at the time you clicked on the Printer icon. If the information you desire is in a scrolling field, and it will not all fit in the window at the same time, you can print multiple views of the window to get all the information you want.

In the future, you will have an option of printing just the information in the current window or else a summary of all the current features, criteria, and weights being used, along with selection recommendations, if any have been given.



The "Disk" icon is pictured at the right.



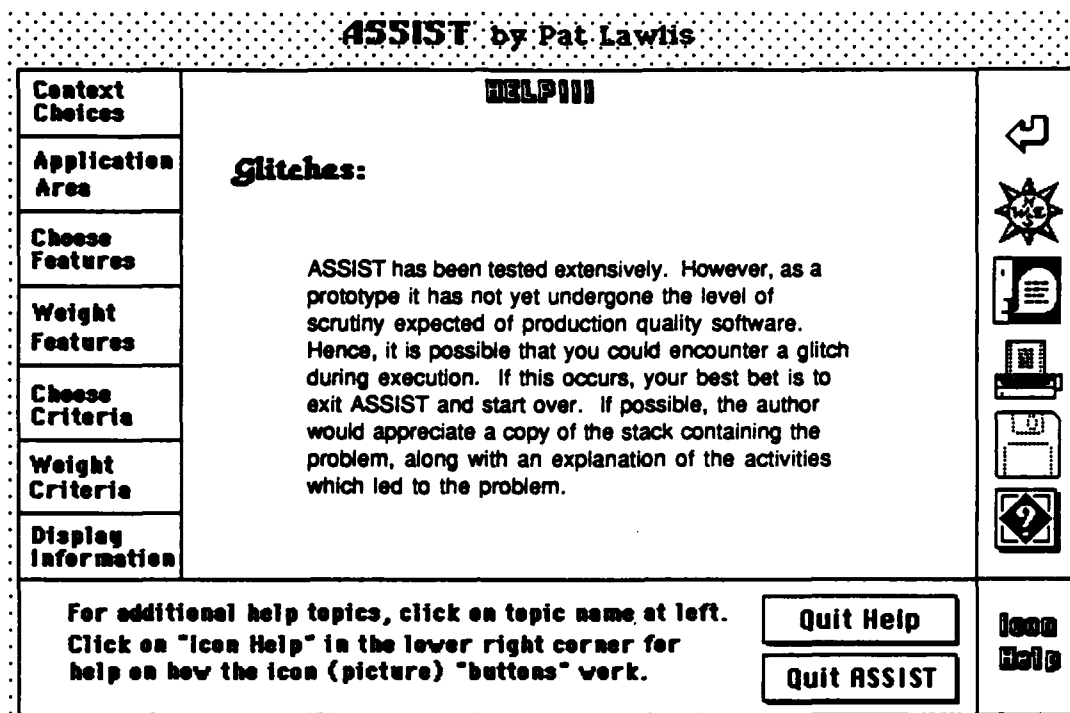
Clicking on the Disk icon will get you to the window shown above. You will have the chance to save the current features, criteria, and weights for a future session. You will also be able to retrieve a previously saved set-up.

If you click on "Save", your choices will automatically be saved to a stack (file) called "Save Stack". This Save Stack could then be moved to another directory or another disk, if desired, for safe keeping. If it stays in the same directory with the ASSIST stack, it will be overwritten the next time a Save is done.

If you click on "Retrieve", the choices will automatically be retrieved from a stack (file) called "Save Stack" which must be in the same directory as the ASSIST stack.

In the future, more flexibility will be provided for the name and location of the Save Stack.

It may seem that only one session with this program will be necessary. However, if you see that some important evaluation data is not in the database, you may want to collect that information, get it added to the database, and then return to the program. You may also want to use very similar specifications for the selection of several different types of software, or you may wish to perform sensitivity analyses using similar specifications.



The "Help" icon is pictured at the right.



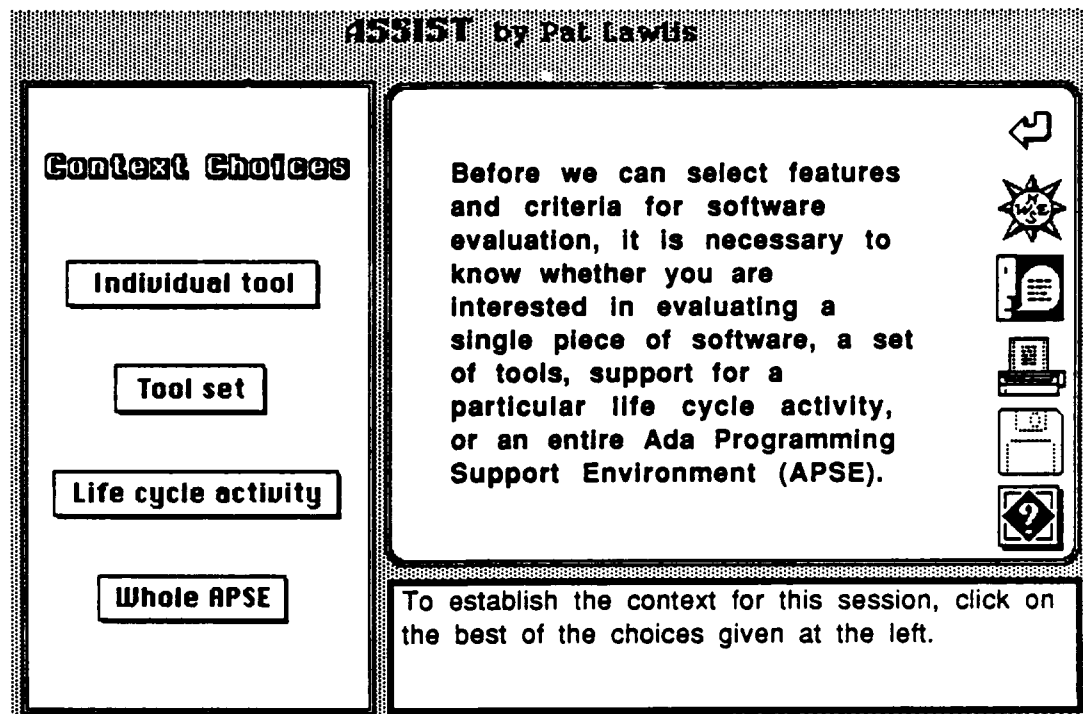
Clicking on the Help icon will bring you to a Help window like the one shown above. Here you will get an explanation of the purpose of the window you were viewing and what you may do while it is displayed. You will be able to get more information on other related parts of the program as well. Click on the Return icon to get back to where you were.

Note that the boxes on the left of the Help window contain the names of the various steps used in executing ASSIST. Each of these is a button, and clicking on a name will take you to a Help window explaining more about that activity.

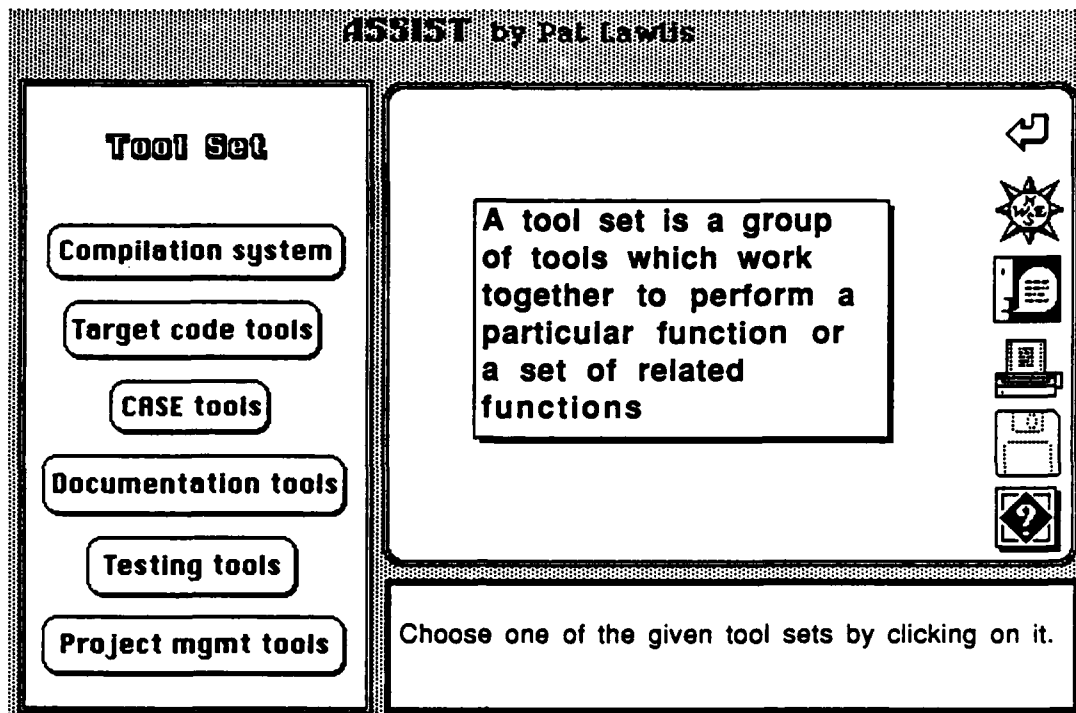
The icons on the right are the six system functions described in this section, and they work the same from the Help window as they do from any other window. The only exception is the "Icon Help" window which is accessed by the button in the bottom right corner. The Icon Help window allows you to get more information about the icons.

Additional buttons may also be present on any particular Help window. Their names imply the type of information they will give you.

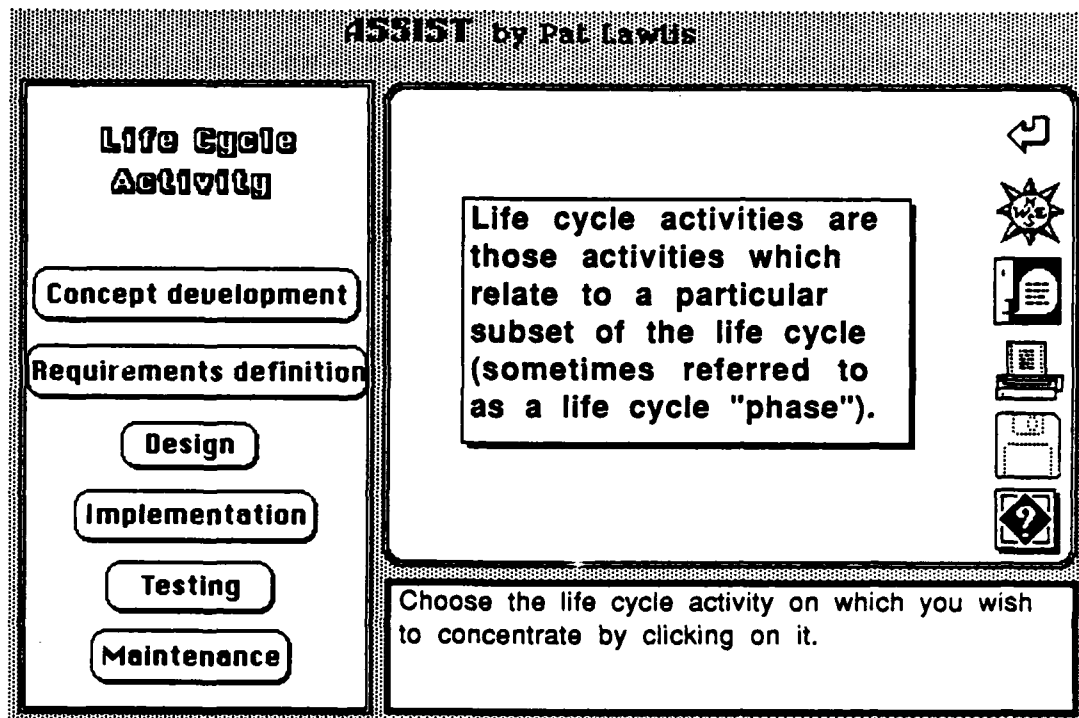
Since you may look at any number of Help windows before returning to a regular window, the "Quit Help" button at the bottom gives you the chance to go directly back to the window from which you originally came without using the Return button multiple times. Of course, "Quit ASSIST" permits you to quit the program at any time.



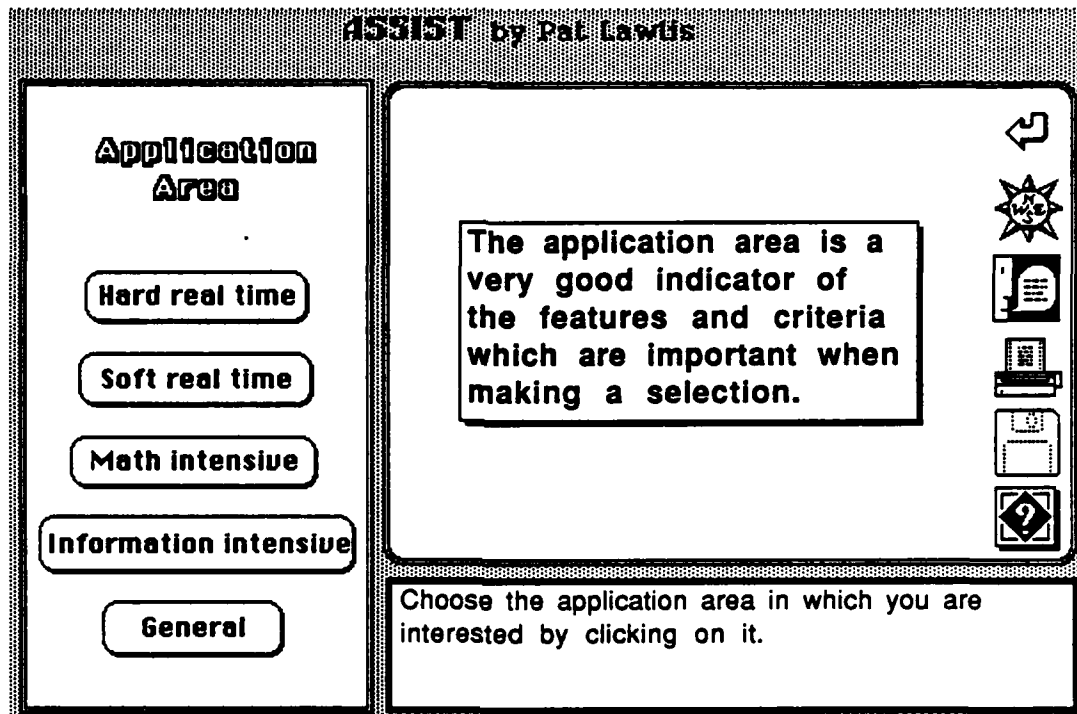
The purpose of choosing a context is to establish the type of software about which ASSIST will make recommendations. In the future, many more context choices will be possible. However, at present only "Tool set" and "Life cycle activity" are possible, and either of these choices will result in the selection of a compilation system.



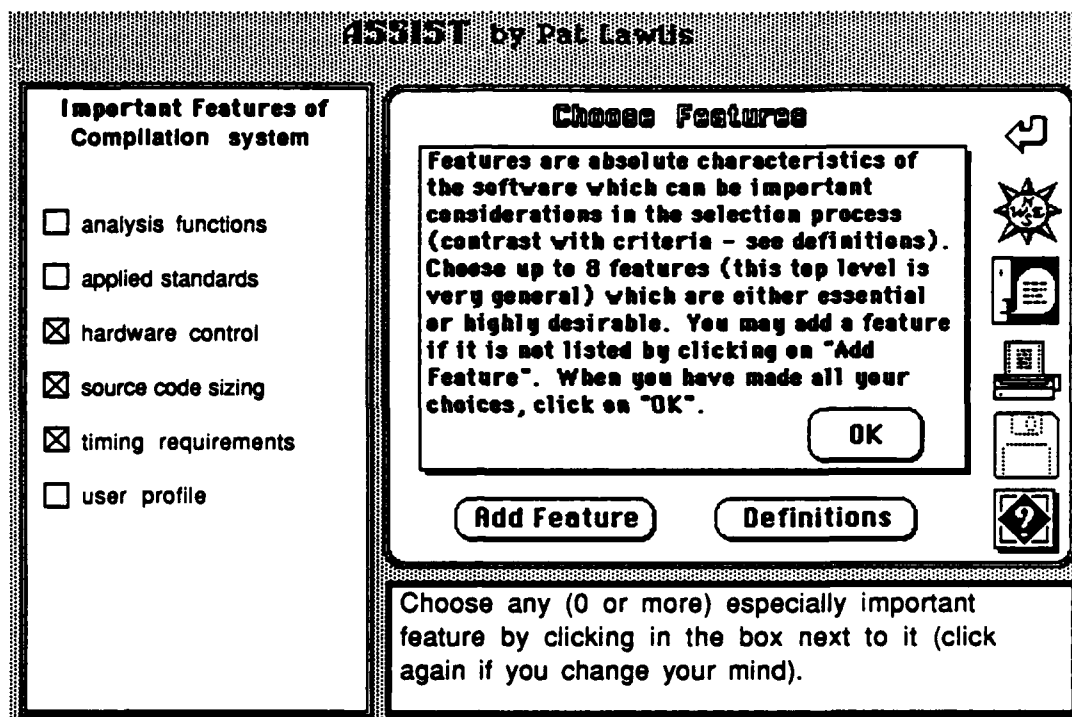
Having this window come into view indicates that your first context choice was "Tool set". Now the purpose of this window is to narrow down the context to determine the type of tool set about which you must make a selection decision. The 6 given tool sets are not necessarily all that are possible, but currently they are more than enough. The only button which is currently active is the one for "Compilation system", and the other 5 will simply give you a message telling you to select compilation system instead. Once you have clicked on the compilation system button, the "Application area" window will come into view.



Having this window come into view indicates that your first context choice was "Life cycle activity". Now the purpose of this window is to narrow down the context to determine the type of life cycle activity about which you must make a selection decision. The 6 given activities are not necessarily all that are possible, but they cover all the activities generally considered to be a part of software development. The "Implementation" button is the only one which is currently active, and the other 5 will simply give you a message telling you to select implementation instead. Once you have clicked on the implementation button, you will see a message indicating that the implementation activity software to be considered for this selection decision will be compilation system software only (because of current program limitations). The "Application area" window will then come into view.



The application area is the last of the windows dealing with narrowing down the type of software for which a selection decision must be made. Of the 5 areas from which to choose, "General" is intended to cover any area which is not covered by the other 4. Any of the 5 application areas may be chosen by clicking on it. Once an area is chosen, you will be ready to begin to choose features and criteria to use in the software evaluation process. The application area you have chosen will determine the features and criteria, along with their corresponding weights, which will be recommended for specification in the software evaluation. Once you have clicked on your choice of area, the "Choose features" window will come into view.



Any number of features may be chosen from those listed in the "Choose Features" window. Once all choices are made from these general features, a click on "OK" takes you to the windows for specifying details for each chosen feature (as long as you choose at least one feature with available details). Weights will be assigned to each feature after all details have been specified.

For a definition of any feature or criterion (as well as definitions for "features" and "criteria"), click on "Definitions" (see page 26). There are no universally accepted definitions for the terms used for features and criteria, so be sure you understand how the terms are used by ASSIST. For a complete glossary of all these terms, see page 29.

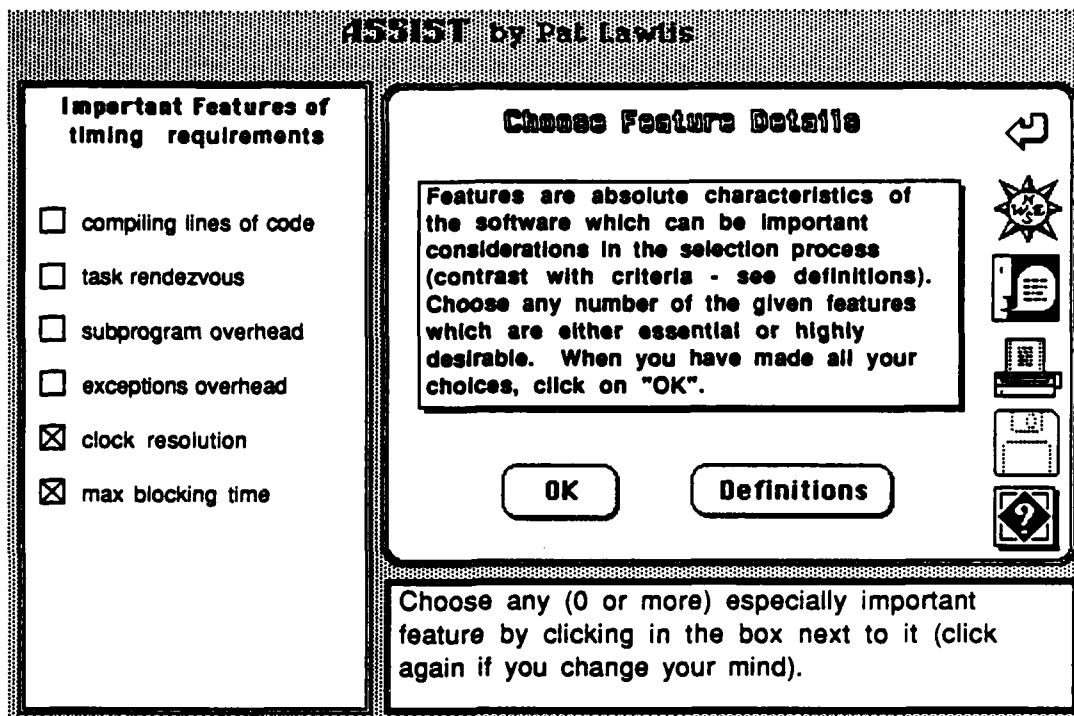
To get a list of other features which may be added, click on "Add Feature". Choose one from this list by clicking on the feature name. If the general list of features is already full when you want to add another, you can still click on "Add Feature". Your new choice will replace an unchosen member of the list. For a look at the hierarchy of feature choices available in ASSIST, see page 24.

If you wish to change a choice, click on the check box to unchoose it. If the check box is covered by a magnifying glass, click on the glass to get to the detail window and unchoose all the details. This will also unchoose the general feature.

After you have chosen one set of features and you have clicked on "Done", you may go back and change features, but you must specify all detail features and weights over again.

For an explanation of the meaning of $\geq$ and $\leq$ when specifying numerical values, see page 27.

For more information on feature choices, see the next page.



This window is used for specifying the details of the general features you have chosen as important. Click on "OK" when you have chosen those details of importance.

Once again, you may refer to the definition of any feature or criterion (as well as the definitions for "features" and "criteria") by clicking on "Definitions" (see page 26). You may also refer to the glossary on page 29.

The Return and Compass buttons will be disabled while you are in the middle of specifying detail features. If detail processing could be interrupted or terminated, system integrity would be in jeopardy.

If you change your mind about specifying certain features while in the middle of detail processing, you can choose none in any window. Once detail processing is completed, you will have ample opportunity to make any desired changes.

If you wish to get a new list of features you will have to unchoose each feature already chosen. If a magnifying glass is covering a check box, you must click on the glass to get to the detail window and then unchoose the details. When you accept the list with no details chosen, the general feature will be unchosen as well. Once the general feature is unchosen, choose it again to get the full list of detail choices again.

Although you may choose any number of features which are listed in a window, there is a limit of 8 which can be listed at any one time. Hence, you may not choose more than 8 in any one window. However, choosing up to the 8 most important features from any particular list should be sufficient to help you arrive at a sound software selection decision.

ASSIST by Pat Lawtis

Weights of Features for Compilation system		Weight Features	
analysis functions	5	Suggested weights are given for each of the features chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK". 1 Marginally important 5 Moderately important 10 Critically important (essential) OK Definitions	
applied standards	5	↶ ☀ 📄 🖨 💾 ❓	
source code sizing	5		
timing requirements	10		
user profile	8		
		For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.	

Suggested weights are given for each of the chosen features. These weights range from 1 to 10, as shown in the window. If you do not wish to change any of these, just click on "OK".

If you wish to change the weight for a feature, click on the weight. This will produce a dialog box which will give you the opportunity to type in the weight you wish to use. Once the weights are as you want them, click on "OK". THE WEIGHTS WILL NOT TAKE EFFECT UNTIL YOU HAVE CLICKED ON THE "OK" BUTTON.

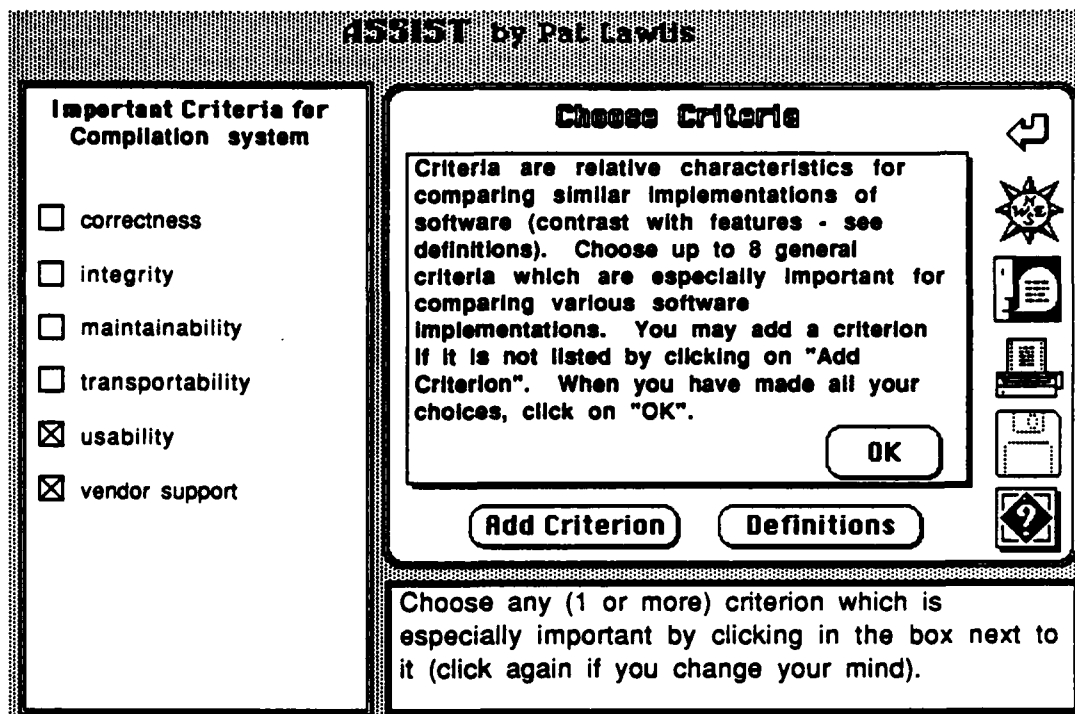
Once again, you may refer to the definition of any feature or criterion (as well as the definitions for "features" and "criteria") by clicking on "Definitions" (see page 26). You may also refer to the glossary on page 29.

ASSIST by Pat Lawlis

Weights of Features for user profile	Weight Feature Details
training 5 moderate skill level 5 intermediate 	Suggested weights are given for each of the features chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK". 1 5 10 Marginally important Moderately important Critically important (essential) OK Definitions For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

The feature details are weighted in the same manner as the general features. However, a weight of 10 for one of these details indicates that this detail feature is absolutely essential to any software you will be selecting. Hence, only software which contains this detail feature will be considered by ASSIST. If a detail feature is very important, but you still do not want to exclude software which does not contain this feature, then weight it at 8 or 9 rather than 10.

The Return and Compass buttons will be disabled while you are in the middle of specifying detail weights. If detail processing could be interrupted or terminated, system integrity would be in jeopardy.



Any number of criteria may be chosen from those listed in the "Choose Criteria" window. Once all choices are made from these general criteria, a click on "OK" takes you to the windows for specifying details for each chosen feature if you choose to specify details. Weights will be assigned to each criterion after all details have been specified.

For a definition of any feature or criterion (as well as definitions for "features" and "criteria"), click on "Definitions" (see page 26). There are no universally accepted definitions for the terms used for features and criteria, so be sure you understand how the terms are used by ASSIST. For a complete glossary of all these terms, see page 29.

To get a list of other criteria which may be added, click on "Add Criterion". Choose one from this list by clicking on the criterion name. If the general list of criteria is already full when you want to add another, you can still click on "Add Criterion". Your new choice will replace an unchosen member of the list. For a look at the hierarchy of criteria choices available in ASSIST, see page 25.

If you wish to change a choice, click on the check box to unchoose it. If the check box is covered by a magnifying glass, click on the glass to get to the detail window and unchoose all the details. This will also unchoose the general criterion.

After you have chosen one set of criteria and have clicked on "Done", you may still go back and change criteria, but you must specify all detail criteria and weights over again.

For more information on criteria choices, see the next page.

ASSIST by Pat Lawlis

Important Criteria for efficiency ☒ processing effectiveness ☒ storage effectiveness	Choose Criteria Details Criteria are relative characteristics for comparing similar implementations of software (contrast with features - see definitions). Choose any number of the given criteria which are especially important for comparing various software implementations. When you have made all your choices, click on "OK". OK Definitions Choose any (1 or more) criterion which is especially important by clicking in the box next to it (click again if you change your mind).

This window is used for specifying the details of the general criteria you have chosen as important. Click on "OK" when you have chosen those details of importance.

Once again, you may refer to the definition of any feature or criterion (as well as the definitions for "features" and "criteria") by clicking on "Definitions" (see page 26). You may also refer to the glossary on page 29.

The Return and Compass buttons will be disabled while you are in the middle of specifying detail criteria. If detail processing could be interrupted or terminated, system integrity would be in jeopardy.

If you change your mind about specifying certain criteria while in the middle of detail processing, you can choose none in any window. Once detail processing is completed, you will have ample opportunity to make any desired changes.

If you wish to get a new list of criteria you will have to unchoose each criterion already chosen. If a magnifying glass is covering a check box, you must click on the glass to get to the detail window and then unchoose the details. When you accept the list with no details chosen, the general criterion will be unchosen as well. Once the general criterion is unchosen, choose it again to get the full list of detail choices again.

Although you may choose any number of criteria which are listed in a window, there is a limit of 8 which can be listed at any one time. Hence, you may not choose more than 8 in any one window. However, choosing up to the 8 most important criteria from any particular list should be sufficient to help you arrive at a sound software selection decision.

ASSIST by Pat Lawlis

Weights of Criteria for Compilation system <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 60%;">correctness</td> <td style="text-align: right;">8</td> </tr> <tr> <td>usability</td> <td style="text-align: right;">10</td> </tr> <tr> <td>vendor support</td> <td style="text-align: right;">8</td> </tr> </table>	correctness	8	usability	10	vendor support	8	Weight Criteria Suggested weights are given for each of the criteria chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK". <table style="margin: auto;"> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">5</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;"> </td> <td style="text-align: center;"> </td> <td style="text-align: center;"> </td> </tr> <tr> <td style="text-align: center;">Marginally important</td> <td style="text-align: center;">Moderately important</td> <td style="text-align: center;">Critically important</td> </tr> </table> <table style="margin: auto;"> <tr> <td style="border: 1px solid black; padding: 5px 20px;">OK</td> <td style="border: 1px solid black; padding: 5px 20px;">Definitions</td> </tr> </table> For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.	1	5	10				Marginally important	Moderately important	Critically important	OK	Definitions
correctness	8																	
usability	10																	
vendor support	8																	
1	5	10																
Marginally important	Moderately important	Critically important																
OK	Definitions																	

Suggested weights are given for each of the chosen criteria. If you do not wish to change any of these, just click on "OK".

If you wish to change the weight for a criterion, click on the weight. This will produce a dialog box which will give you the opportunity to type in the weight you wish to use. Once the weights are as you want them, click on "OK". THE WEIGHTS WILL NOT TAKE EFFECT UNTIL YOU HAVE CLICKED ON THE "OK" BUTTON.

Once again, you may refer to the definition of any feature or criterion (as well as the definitions for "features" and "criteria") by clicking on "Definitions" (see page 26). You may also refer to the glossary on page 29.

ASSIST by Pat Lawlis

Weights of Criteria for efficiency

processing effectiveness	8
storage effectiveness	5

Weight Criteria Details

Suggested weights are given for each of the criteria chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK".

1
5
10

Marginally
important

Moderately
important

Critically
important

OK

Definitions








For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

The criteria details are weighted in the same manner as the general criteria. A weight of 10 has no special meaning as it has for feature details because criteria are relative rather than absolute characteristics. A weight of 10 for a detail criteria simply means it is critically important for the software to be selected.

The Return and Compass buttons will be disabled while you are in the middle of specifying detail weights. If detail processing could be interrupted or terminated, system integrity would be in jeopardy.

Features in Database

analysis functions

- consistency checking
- cross referencing
- data flow analysis
- mutation analysis
- regression testing
- requirements simulation
- statistical profiling
- traceability analysis

applied standards

- Ada (MIL-STD-1815A)
- CAIS (MIL-STD-1838A)
- PCTE
- DIANA
- GKS
- PHIGS
- DoD-STD-2167A

associated tool requirements

configuration requirements

- host hardware
- target hardware
- host memory needed
- host disk capacity needed
- peripheral devices
- operating system
- support software
- distributed system

contractual matters

- no restrictions on users
- number of users
- number of CPUs
- sale of derived software
- source code available
- support available

cost

hardware control

user profile

- skill level
- training

management functions

- configuration management
- cost management
- object management
- performance monitoring
- program library management
- quality management
- resource management

numerics

- bits in integer
- max integer
- bits in float
- bits in exponent
- fixed point delta
- digits in float
- long rep forms
- short rep forms

options

security issues

source code sizing

- lines in unit
- units in compile
- entries in task
- elements in aggregate
- discriminants in record
- alternatives in case
- alternatives in select
- instantiations of generic

timing requirements

- compiling lines of code
- task rendezvous
- subprogram overhead
- exceptions overhead
- clock resolution
- max blocking time

transformation functions

- incremental compilation
- editing
- formatting
- linking/loading
- activities transformation
- object transformation
- program generation

Criteria in Database

correctness

- completeness
- consistency
- traceability

efficiency

- communication effectiveness
- processing effectiveness
- storage effectiveness

expandability

- augmentability
- generality
- modularity
- self documentation
- simplicity

integrity

- security
- standards compatibility

interoperability

- communication commonality
- data commonality
- modularity
- rehostability
- retargetability

maintainability

- augmentability
- communicativeness
- consistency
- modularity
- self documentation
- simplicity
- structuredness
- test availability

reliability

- accuracy
- completeness
- consistency
- fault tolerance
- modularity
- simplicity

reusability

- application independence
- generality
- hardware independence
- modularity
- operating system independence
- self documentation

survivability

- autonomy
- distributedness
- fault tolerance
- modularity
- reconfigurability

transportability

- hardware independence
- modularity
- operating system independence
- rehostability
- retargetability
- self documentation
- support software independence

usability

- capacity
- ease of installation
- ease of use
- maturity
- on-line help
- power
- tailorability
- user documentation

vendor support

- corporate health
- pricing policies
- reputation
- support policies

verifiability

- communicativeness
- modularity
- self documentation
- simplicity
- standards compatibility
- structuredness
- test availability

ASSIST by Pat Lawlis**Definitions**

Features are characteristics of software which are used to specify absolute requirements for software implementations.

Criteria are characteristics of software which are used to make relative comparisons of similar software implementations.

Features

activities transformation
Ada (MIL-STD-1815A)
alternatives in case
alternatives in select
analysis functions
applied standards
associated tool reqmts
bits in float
bits in exponent
bits in integer
CAIS (MIL-STD-1838A)
clock resolution
compiling lines of code
configuration management

Scroll through either the Features or Criteria (drag the box or click on the arrows) until you find the term to be defined. Then click on the term. Click on the return arrow when done.

**Criteria**

accuracy
application independence
augmentability
autonomy
capacity
communication commonality
communication effectiveness
communicativeness
completeness
consistency
corporate health
correctness
data commonality
distributedness

You can get to this window from any of the windows used for specifying features and criteria. Any number of definitions may be examined while viewing this window. Just find a term you want to have defined and then click on it.

There are no universally accepted definitions for the terms used for features and criteria, so be sure you understand how the terms are used in ASSIST. For a complete glossary of all these terms, see page 29.

Specifying Numerical Values

When you are asked to specify a numerical value for a feature, the value you are to provide is the smallest acceptable number (if the sign given is $\geq$) or the largest acceptable number (if the sign given is $\leq$). Be sure you know the definition of the feature as it is used in ASSIST. See the glossary on page 29 if necessary. The examples below should help to clarify this matter.

Example 1:

You are given the statement:

Specify number of users as $\geq$

some number, where you can change the number or accept it as is.

You are to specify the smallest acceptable number of users who may simultaneously execute a single purchased copy of the software (see the definition of number of users in the Glossary, page 29). If you were to specify 4, it means that an acceptable software product would have to permit at least 4 simultaneous users. On the other hand, if you were to specify 1, then any number of users is acceptable.

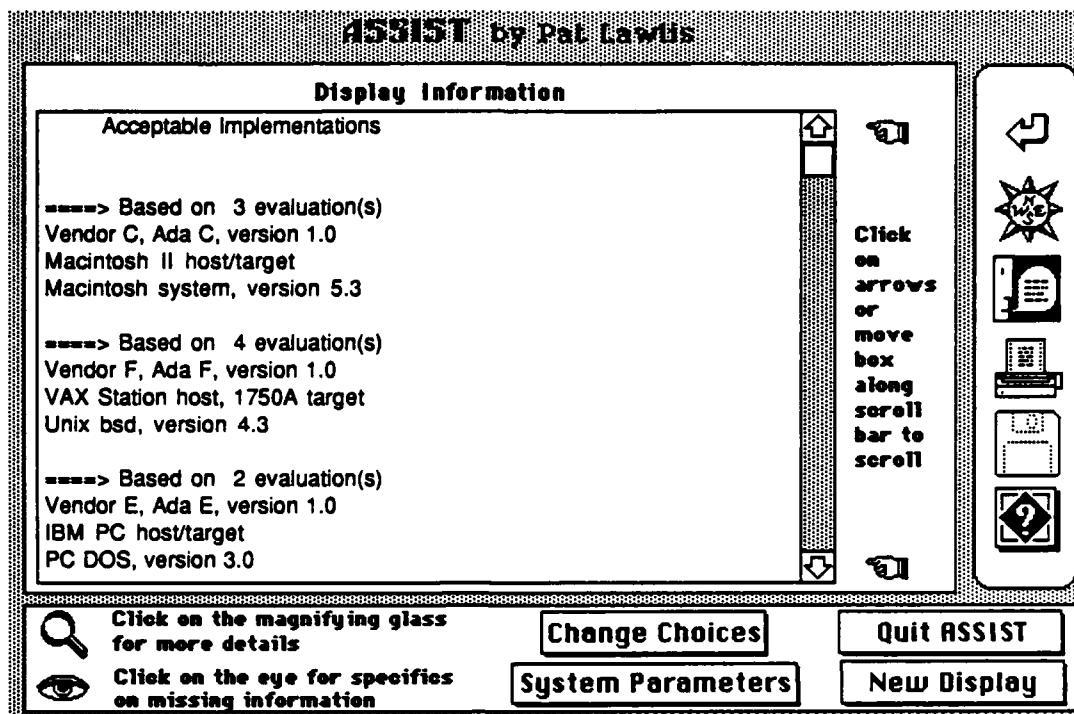
Example 2:

You are given the statement:

Specify host disk capacity needed as $\leq$ (MB)

some number, where you can change the number or accept it as is.

You are to specify the largest acceptable size (in megabytes) for the combined storage required for the software to run (see the definition of host disk capacity needed in the Glossary, page 29). If you were to specify 4, it means that the software should require no more than 4 megabytes of host disk storage in order to run. You should specify the largest amount of storage space you are willing to dedicate to this software.



The main display window shows the product identification of all software products in the current database of the type under consideration. There are 3 possible categories which are listed. The first list is of those products which are acceptable according to the features, criteria, and weights specified. The second is of those which are unacceptable, and the third is of those which were not even considered because they did not have the detail features which were identified as absolutely required (with a weight of 10).

The detail windows give specifics on the calculations used to determine the acceptable and unacceptable products. Click on the magnifying glass to see the formulas used in the calculations, as well as the results of the calculations. Click on the eye to see the chosen features and criteria for which there is no data in the database for a product. Click on "System Parameters" to see the parameters used in the calculations. These parameters may be changed if desired.

Click on "Change Choices" if you want to make some changes to the chosen features or criteria or their weights. This will take you back to the "Choose Features" window. If you want to start over from scratch instead (for a different software product), click on the compass and go back to "Context Choices".

Click on "New Display" if you have already made some changes (to the parameters, features, criteria, or weights), and you now want new recommendations based on these changes.

Glossary

The following definitions have been adapted from several sources [see P. K. Lawlis, *Supporting Selection Decisions Based on the Technical Evaluation of Ada Environments and Their Components*, PhD dissertation, Arizona State University, 1989]. In cases where these sources provided different definitions for the same term, all definitions have been included. Each definition is given in one sentence. The first definition always expresses the sense in which the term is used in ASSIST. The definitions for features (absolute characteristics) are preceded by (f) and the definitions for criteria (relative characteristics) are preceded by (c).

accuracy - (c) A quantitative measure of the magnitude of error expressed as a function of the relative error, with a high value corresponding to a small error. The precision of computations and control. Those characteristics of software which provide the required precision in calculations and outputs.

activities transformation - (f) A software function which performs a transformation on a product of one life cycle activity to produce a product for another activity.

Ada (MIL-STD-1815A) - (f) The standard which specifies the Ada language.

alternatives in case - (f) The maximum number of individual alternatives which can be defined in a case statement.

alternatives in select - (f) The maximum number of alternatives which can be defined in a select statement.

analysis functions - (f) Software functions which provide an examination of a substantial whole to determine both qualitative and quantitative properties.

application independence - (c) The extent to which software is not dependent on the support required for a particular application. Those characteristics of software which determine

User Manual 30

its nondependency on database system, microcode, computer architecture, and algorithms.

applied standards - (f) Standards to which software or its inputs or outputs conform.

associated tool requirements - (f) Tools which must be available and compatible with the software.

augmentability - (c) The extent to which software provides for expansion of capability for functions and data. Those characteristics of software which provide for expansion of capability for functions and data.

autonomy - (c) The extent to which software is not dependent on interfaces and functions. Those characteristics of software which determine its non-dependency on interfaces and functions.

bits in float - (f) The total number of bits used for a float representation.

bits in exponent - (f) The number of bits used for the representation of the exponent (including its sign) in a float representation.

bits in integer - (f) The number of bits used for an integer representation.

CAIS (MIL-STD-1838A) - (f) The standard which specifies the Common APSE Interface Set, a set of interfaces to the APSE kernel.

capacity - (c) The extent of the upper and lower limits of the functions implemented by a tool.

clock resolution - (f) The amount of time distinguishing (the difference between) two consecutive clock times.

communication commonality - (c) The degree to which standard interfaces, protocols, and bandwidths are used. Those

User Manual 31

characteristics of software which provide for the use of interface standards for protocols, routines, and data representations.

communication effectiveness - (c) The extent to which software performs its intended functions with a minimum consumption of communications resources. Those characteristics of the software which provide for minimum utilization of communications resources in performing functions.

communicativeness - (c) The degree to which the program provides feedback while it is operating to keep the user informed of the functions being performed.

compiling lines of code - (f) The number of lines of source code which are compiled in a minute (wall clock time).

completeness - (c) The extent to which a component provides the complete set of operations necessary to perform a function. The degree to which full implementation of required function has been achieved. Those characteristics of software which provide full implementation of the functions required.

configuration management - (f) A software function which establishes baselines for configuration items, controls the changes to these baselines, and controls releases to the operational environment.

configuration requirements - (f) Those specific components of system hardware and/or software which are required in order for the software to function correctly.

consistency - (c) The extent to which uniform design and documentation techniques have been used throughout the software development project. The use of uniform design and documentation techniques throughout the software development project. Those characteristics of software which provide for uniform design and implementation techniques and notation.

User Manual 32

consistency checking - (f) A software function which determines whether or not an entity is internally consistent in the sense that it is consistent with its specification.

contractual matters - (f) Features determining the legal use of and support provided for software which may be specified in a contract with the vendor at the time of purchase.

corporate health - (c) The extent to which it is reasonable to assume that the vendor will remain in business with the ability to continue the current level of customer support.

correctness - (c) The extent to which software design and implementation conform to specifications and standards. The extent to which a program satisfies its specification and fulfills the customer's mission objectives. The extent to which software is free from design defects and from coding defects; that is, fault free. Agreement between a component's total response and the stated response in the functional specification (functional correctness), and/or between the component as coded and the programming specification (algorithmic correctness).

cost - (f) The total price associated with the purchase and productive use of the software (including the basic software price, training costs, installation costs, and any other ancillary costs associated with making the software a productive part of the user's facility).

cost management - (f) A software function which manages cost functions (such as the cost organization structure and the cost estimation methodology).

criteria - Characteristics of software which are used to make relative comparisons of similar software implementations.

cross referencing - (f) A software function which references entities to other entities by logical means.

data commonality - (c) The extent to which standard data structures and types are used throughout the program. The

User Manual 33

use of standard data structures and types throughout the program. Those characteristics of software which provide for the use of interface standards for data representations.

data flow analysis - (f) A software function which analyzes the formal requirements statements to determine interface consistency and data availability.

DIANA - (f) The standard which specifies a Descriptive Intermediate Attributed Notation for Ada, an abstract data type such that each object of the type is a representation of an intermediate form of an Ada program.

digits in float - (f) The largest number of decimal digits which may be represented by a float.

discriminants in record - (f) The maximum number of discriminants which can be defined for a single record type.

distributed system - (f) A system in which software functions are geographically or logically separated within the system.

distributedness - (c) The degree to which software functions are geographically or logically separated within the system. Those characteristics of software which determine the degree to which software functions are geographically or logically separated within the system.

DoD-STD-2167A - (f) The standard which establishes uniform requirements for software development that are applicable throughout the system life cycle.

ease of installation - (c) The relative ease with which a software product may be integrated into its operational environment and tested in this environment to ensure that it performs as required.

ease of use - (c) The relative ease with which a novice user can become an effective user of the program.

User Manual 34

editing - (f) A software function which provides for selective revision of computer-resident data (the data may be textual, graphical, some internal representation, etc.).

efficiency - (c) The extent to which software performs its intended functions with a minimum consumption of computing resources. The amount of computing resources and code required by a program to perform its function. The relative extent to which a resource is utilized. The ratio of actual utilization of the system resources to optimum utilization.

elements in aggregate - (f) The maximum number of elements which can constitute an aggregate.

entries in task - (f) The maximum number of entries which can be defined in a single task.

exactness - The measure of assuredness that a component does no more than it was specified to do and does not contain malicious code.

exceptions overhead - (f) The execution overhead time which is attributable to the presence of exception handlers in the unit.

expandability (extensibility) - (c) The degree to which architectural, data, or procedural design can be extended. The relative effort to increase the software capability or performance by enhancing current functions or by adding new functions or data. The extent to which a component allows new capabilities to be added and existing capabilities to be easily tailored to user needs.

fault tolerance - (c) The extent to which the system has the built-in capability to provide continued correct execution in the presence of a limited number of hardware or software faults. Those characteristics of software which provide for continuity of operations under and recovery from non-nominal conditions. The protection of a component from itself, user errors, and system errors. The ability to recover and provide meaningful diagnostics in the event of unforeseen situations. The damage that occurs when the program encounters an error.

User Manual 35

- features** - Characteristics of software which are used to specify absolute requirements for software implementations.
- fixed point delta** - (f) The smallest interval which may be used to distinguish among fixed point values.
- formatting** - (f) A software function which arranges data according to predefined and/or user-defined conventions.
- generality** - (c) The breadth of the potential application of program components. Those characteristics of software which provide breadth to the functions performed with respect to the application.
- GKS** - (f) The standard which specifies the Graphical Kernel System, a graphics system which allows programs to support a wide variety of graphics devices and which is defined independently of programming languages.
- hardware control** - (f) The ability of the software to control hardware directly (such as interrupts, bit manipulations, file servers, task scheduling, preemption, etc.).
- hardware independence** - (c) The degree to which the software is decoupled from the hardware on which it operates. Those characteristics of software which determine its nondependency on specific hardware. The degree to which hardware dependencies are isolated in a distinct library unit.
- host disk capacity needed** - (f) The combined storage size (in megabytes) required of the on-line disk units of the host hardware to ensure that the software will run properly.
- host hardware** - (f) The specification of the manufacturer and model of the computer hardware which will serve as the development platform for the software to be developed.
- host memory needed** - (f) The size (in megabytes) required of the primary memory of the host hardware to ensure that the software will run properly.

User Manual 36

incremental compilation - (f) A software function which produces new object code for a particular source code unit from the previous object code for that unit and a set of specified changes to the source code which produced the original object code.

instantiations of generic - (f) The maximum number of times a single generic unit can be instantiated.

integrity - (c) The extent to which unauthorized access to or modification of software or data can be controlled. The extent to which the software will perform without failures due to unauthorized access to the code or data within a specified time period. The probability that the system will perform without failure and will protect the system and data from unauthorized access.

interoperability - (c) The degree to which an APSE may provide data base objects and their relationships in forms usable by the components and user programs of another APSE without conversion. The extent to which two or more systems have the ability to exchange information and to mutually use the information that has been exchanged. The effort required to couple one system to another. The relative effort to couple the software of one system to the software of another system. The probability that two or more systems can exchange information under stated conditions and use the information that has been exchanged.

lines in unit - (f) The maximum number of source code lines which can be compiled in one compilation unit.

linking/loading - (f) A software function which creates a load/executable module on the host machine from one or more independently translated object modules or load modules by resolving cross-references among the object modules, and possibly relocating elements.

long rep forms - (f) The ability to specify a number (integer or float) which will be represented using more total bits than is

User Manual 37

used by numbers of the same base type without the "long" designation.

maintainability - (c) The extent to which a component facilitates updating to satisfy new requirements or to correct deficiencies. The effort required to locate and fix an error in a program. The ease of effort for locating and fixing a software failure within a specified time period. The ease with which software can be maintained. The probability that the system can be restored to a specified condition within a specified amount of time.

malicious code - operations which covertly damage or attempt to by-pass system security.

management functions - (f) Software functions which aid the management or control of system/software development.

maturity - (c) The extent to which a component has been used in the development of deliverable software by typical users and to which the feedback from that use has been reflected in modifications to the component.

max blocking time - (f) The maximum amount of overhead time used by the runtime system to block a task.

max integer - (f) The maximum number which may be represented as an integer.

modularity - (c) The extent to which software is composed of discrete components such that a change to one component has minimal impact on other components. The extent to which a component is implemented in a hierarchical structure in which identifiable functions are isolated in separate compilation units. The functional independence of program components. Those characteristics of software which provide a structure of highly cohesive components with optimum coupling.

mutation analysis - (f) A software function which applies test data to a program and its "mutants" (programs that contain one or more likely errors) in order to determine test data adequacy.

User Manual 38

no restrictions on users - (f) Not disallowing or constraining the use of the software by a particular class of users (such as people not employed by the purchasing organization).

number of CPUs - (f) The total number of computers which may legally serve as the residence for a particular software component.

number of users - (f) The maximum number of users permitted simultaneous execution of a single purchased copy of the software.

numerics - (f) Software features which determine the computational capabilities of the software.

object management - (f) A software function which manages a collection of interrelated data (objects) stored together with controlled redundancy, serving one or more applications and independent of the programs using the data (objects).

object transformation - (f) A software function which performs a transformation on a particular system object to produce another system object.

on-line help - (c) The extent to which user documentation is readily available to the user from the program while it is operating.

operating system - (f) The specification of the name and version of the operating system under which the software will run.

operating system independence - (c) The degree to which the program is independent of operating system characteristics. Those characteristics of software which determine its nondependency on a specific operating system. The degree to which operating system dependencies are isolated in a distinct library unit.

options - (f) Software features whose specified values (each of which causes the software to execute in a somewhat different, yet controlled, manner) are set by the user.

User Manual 39

PCTE - (f) The standard which specifies the Portable Common Tool Environment, a hosting structure defined by a set of program-callable primitives which support the execution of programs in terms of a virtual, machine independent level of comprehensive facilities.

performance monitoring - (f) A software function which monitors the performance characteristics of the finished product.

peripheral devices - (f) The hardware devices which are attached to and work with the computer but are not an integral part of it (such as printers, terminals, etc.).

PHIGS - (f) The standard which specifies the Programmers Hierarchical Interactive Graphics Standard, a sophisticated graphics support system that controls the definition, modification, and display of hierarchical graphics data.

power - (c) The extent to which a component has capabilities, such as default options and wild card operations, that contribute to the effectiveness of the user.

pricing policies - (c) The degree to which the vendor's prices for product support and upgrades are reasonable and in accordance with accepted practice within the software industry.

processing (execution) effectiveness - (c) The extent to which software performs its intended functions with a minimum consumption of processing resources. The run-time performance of a program. Those characteristics of the software which provide for minimum utilization of processing resources in performing functions. The choice between alternative algorithms based on those taking the least amount of time.

program generation - (f) A software function which provides the translation or interpretation used to construct computer programs (such as language translator generator, syntax

User Manual 40

analyzer generator, code generator generator, environment definition generator, user interface generator, etc.).

program library management - (f) A software function which performs the creation, manipulation, display, and deletion of the various components of a program library.

quality management - (f) A software function which manages the determination of the achieved level of quality in deployed software systems.

reconfigurability - (c) The extent to which software provides for continuity of system operation when one or more processor, storage units, or communication links fails. Those characteristics of software which provide for continuity of system operation when one or more processors, storage units, or communication links fails.

regression testing - (f) A software function which performs the rerunning of tests in order to detect errors spawned by changes or corrections made during software development and maintenance.

rehostability - (c) The extent to which an APSE component may be installed on a different host or different operating system with a minimum of reprogramming. The ability of an APSE component to be installed on a different host or different operating system with needed reprogramming localized to the KAPSE or machine dependencies.

reliability - (c) The extent to which a component can be expected to perform its intended functions in a satisfactory manner over a specified period of time. The extent to which a program can be expected to perform its intended function with required precision. The extent to which the software will perform without any failures within a specified time period. The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability that the system will perform as intended under stated conditions for a specified period of time.

User Manual 41

reputation - (c) The degree of confidence expressed by program users in the vendor's willingness and ability to provide support for the program.

requirements simulation - (f) A software function which executes code-enhanced requirements statements to examine functional interfaces and performance.

resource management - (f) A software function which manages the resources attributed to an entity.

retargetability - (c) The extent to which an APSE component may accomplish its function with respect to another target with a minimum of modification. The ability of an APSE component to accomplish its function with respect to another target.

reusability - (c) The extent to which a program (or parts of a program) can be reused in other applications. The relative effort to convert a software component for use in another application. The relative effort to adapt software for use in another application.

sale of derived software - (f) Disallowing or constraining the conditions under which some portion of the purchased software may be included in software provided by the purchaser to a third party.

security - (c) The extent of protection of computer hardware and software from accidental or malicious access, use, modification, destruction, or disclosure. The availability of mechanisms that control or protect programs and data.

security issues - (f) Features which affect the use of the software in a classified environment.

self documentation - (c) The degree to which the source code provides meaningful documentation. Those characteristics of software which provide explanation of the implementation of functions. The technical data, including on-line, documentation, listings, and printouts, which serve the purpose of elaborating the design or details of a component.

User Manual 42

short rep forms - (f) The ability to specify a number (integer or float) which will be represented using fewer total bits than is used by numbers of the same base type without the "short" designation.

simplicity - (c) The extent to which the complexity of a system or system component (determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, the type of data structures, and other system characteristics) is kept to a minimum. The degree to which a program can be understood without difficulty. Those characteristics of software which provide for definition and implementation of functions in the most noncomplex and understandable manner.

skill level - (f) The level of experience in using similar software.

source code available - (f) The possibility that the source code of the software can be purchase.

source code sizing - (f) The limits imposed on the size of selected components of the software.

standards compatibility - (c) The degree to which the program conforms to specific standards.

statistical profiling - (f) A software function which provides the analysis of a program to determine statement types, number of occurrences of each statement type, and the percentage of each statement type in relation to the complete program.

storage effectiveness - (c) The extent to which software performs its intended functions with a minimum consumption of storage resources. Those characteristics of the software which provide for minimum utilization of storage resources. The choice between alternative source code constructions based on those taking the minimum number of words of object code or in which the information-packing is high.

User Manual 43

structuredness - (c) The degree to which the program is constructed of a basic set of control structures, each one having one entry point and one exit.

subprogram overhead - (f) The overhead time involved in executing a subprogram call.

support available - (f) The possibility of purchasing support for the software from the vendor on a continuing basis.

support policies - (c) The type and extent of support provided by the vendor for the software.

support software - (f) The specification of the name and version of the support software required to work with the software in question to ensure proper functionality.

support software independence - (c) The degree to which the program is independent of nonstandard programming language features and other environmental constraints. Those characteristics of software which determine its nondependency on specific support software in the environment (utilities, input and output routines, libraries). The degree to which support software dependencies are isolated in a distinct library unit.

survivability - (c) The extent to which the software will perform and support critical functions without failures within a specified time period when a portion of the system is inoperable. The extent to which software will continue performing when a portion of the system has failed.

tailorability - (c) The extent to which the user interface of the program may be altered to conform to the preferences of the user.

target hardware - (f) The specification of the manufacturer and model of the computer hardware on which the software to be developed will be executed.

task rendezvous - (f) The overhead time required to accomplish a task rendezvous.

User Manual 44

test availability - (c) The extent to which tests are available to verify that a program functions in accordance with its requirements. The extent to which tests are available to support the evaluation of a program's performance with respect to specific verification criteria.

timing requirements - (f) The limits imposed on the execution time of selected components of the software.

traceability - (c) The ability to trace a design representation or actual program component back to requirements. Those characteristics of software which provide a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment.

traceability analysis - (f) A software function which checks for internal consistency within the software requirements specification.

training - (f) The amount of training required to be able to use the software productively.

transformation functions - (f) Software functions which describe how the subject is manipulated to accommodate the user's needs.

transportability (portability) - (c) The effort required to transfer the program from one hardware and/or software system environment to another. The relative effort to transport the software for use in another environment. The extent to which a component can be adapted for use in another environment. The extent to which a component may be installed on a different APSE without change in functionality.

units in compile - (f) The largest number of compilation units which can be involved in a single compile.

usability - (c) The extent to which resources required to acquire, install, learn, operate, prepare input for, and interpret output

User Manual 45

of a component are minimized. The effort required to learn, operate, prepare input, and interpret the output of a program. The relative effort for using software (training and operation). The probability that users can operate the system under specified conditions without user error given they have received specified training.

user documentation - (c) The extent to which documentation conveys to the end user of a system instructions for using the system to obtain desired results. The technical data which serve the purpose of elaborating the design or details of a component to the user.

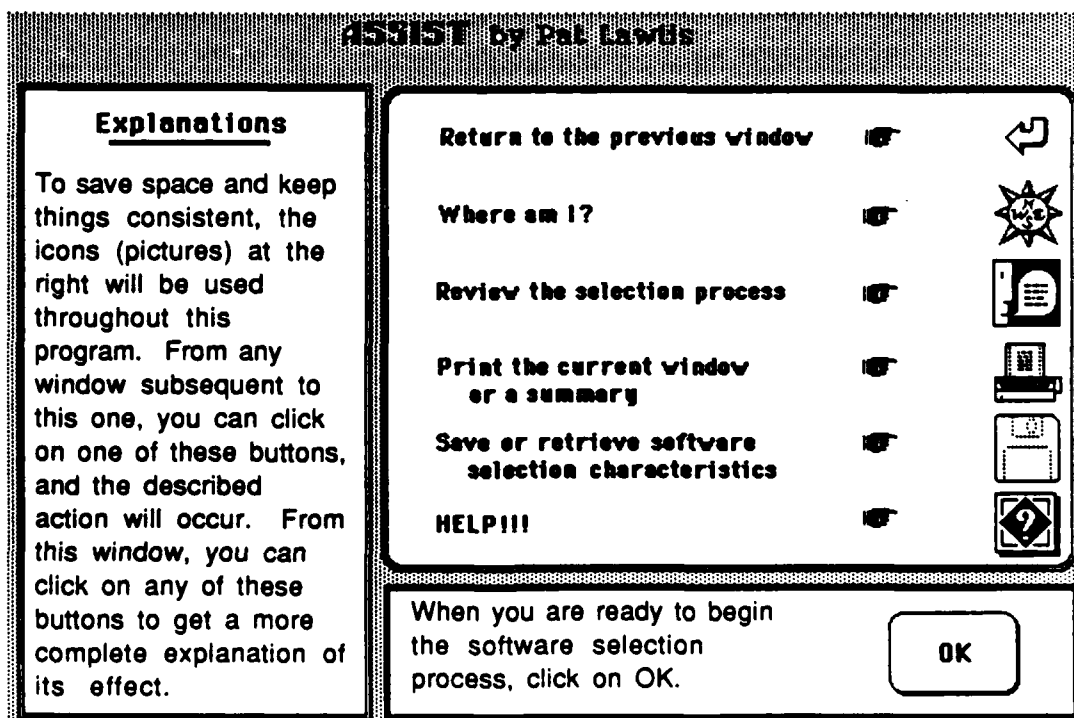
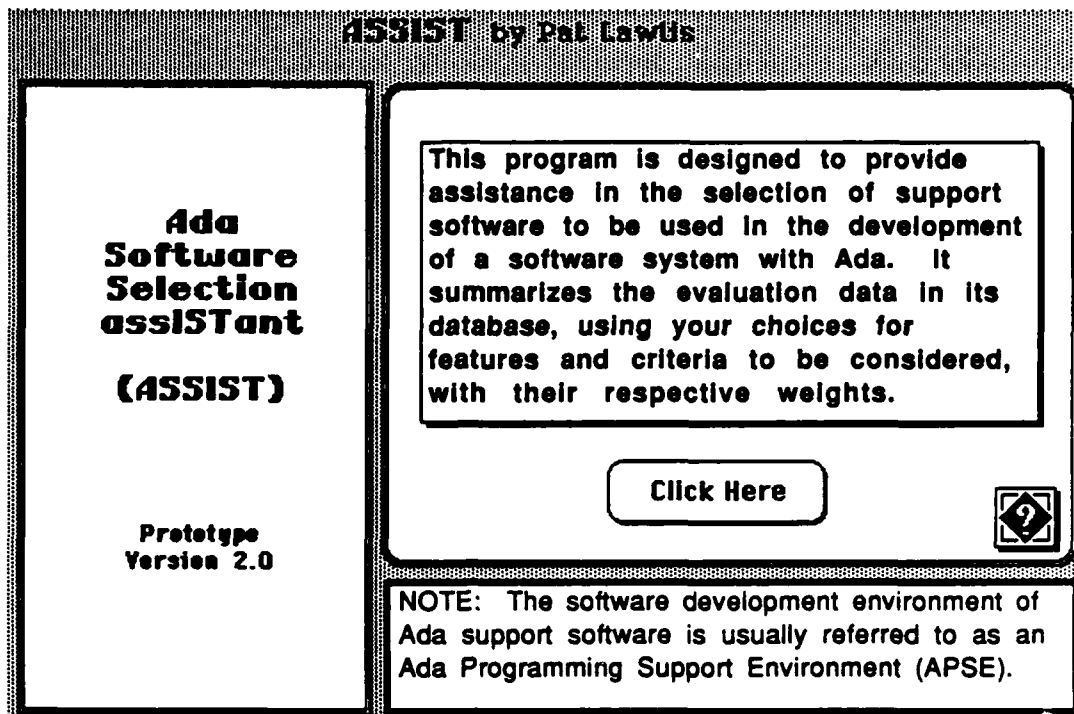
user profile - (f) Characteristics required of the user in order to use the software productively.

vendor support - (c) The extent to which a vendor is willing and able to provide the software user with assistance to ensure that the software performs desired functions and is willing and able to support the continuing maturation of the product.

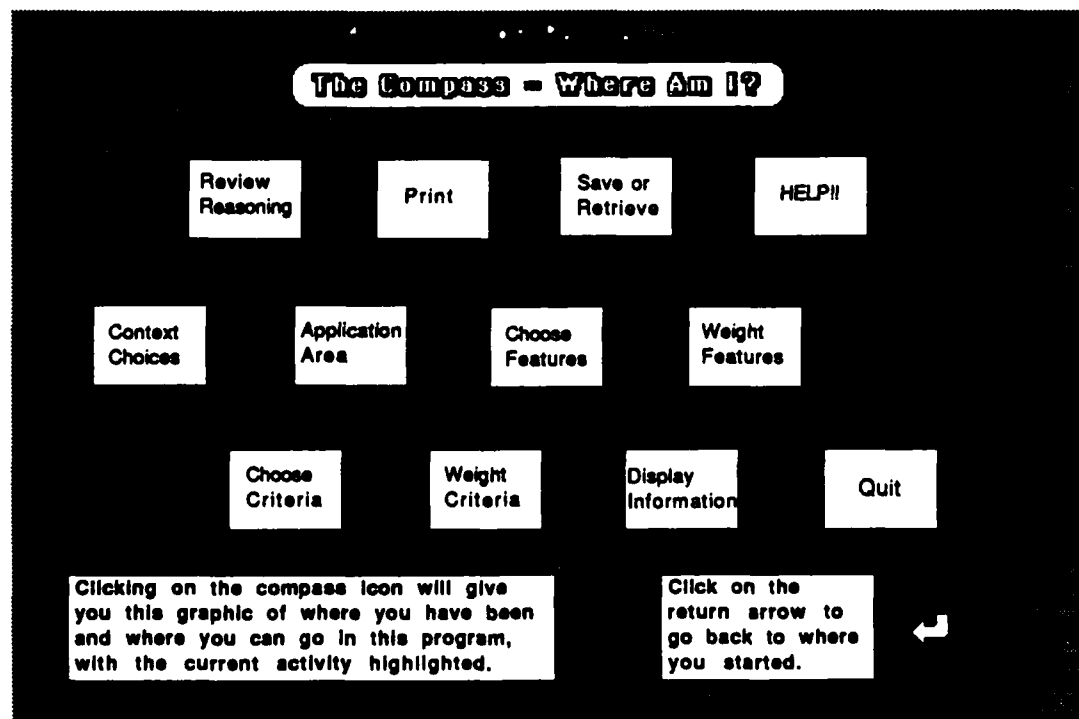
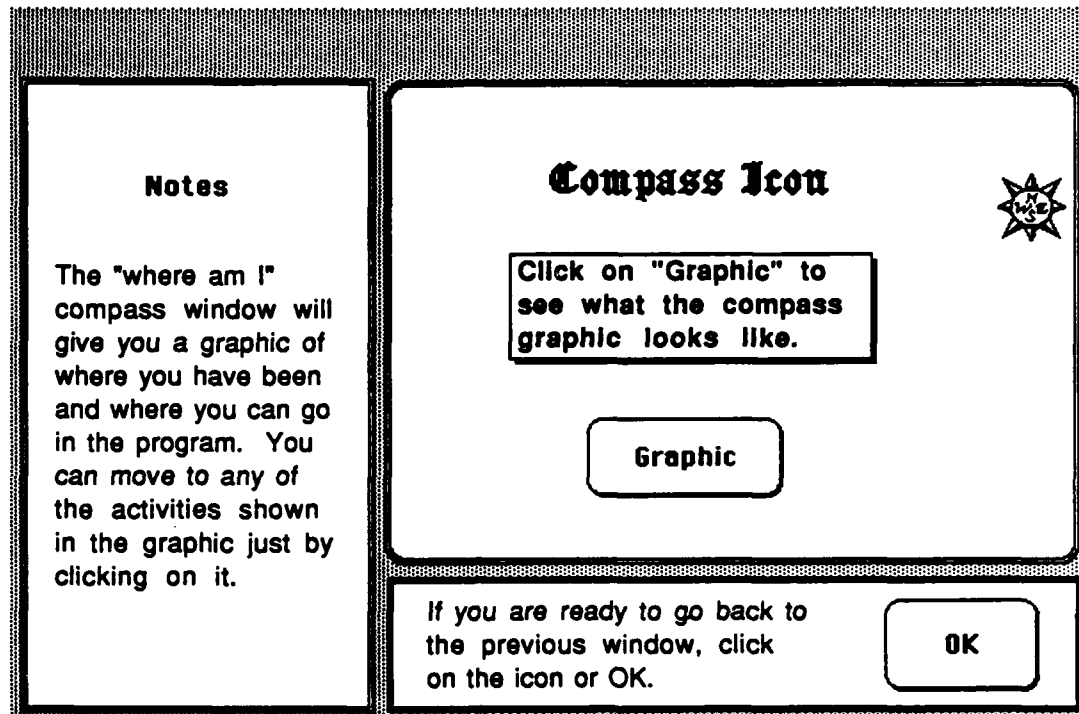
verifiability - (c) The extent to which a component facilitates the establishment of verification criteria and supports evaluation of its performance. The effort required to test a program to ensure that it performs its intended function. The relative effort to verify the specified software operation and performance. The extent to which the specified system operation and performance determine the conditions and criteria for tests. The extent to which a component facilitates the evaluation of its correctness, completeness, and exactness.

Appendix M

Windows Seen In Management-Oriented Scenario



Management-Oriented Scenario 2



Management-Oriented Scenario 3

4531ST by Pat Lawlis

Context Choices

Individual tool

Tool set

Life cycle activity

Whole APSE

Before we can select features and criteria for software evaluation, it is necessary to know whether you are interested in evaluating a single piece of software, a set of tools, support for a particular life cycle activity, or an entire Ada Programming Support Environment (APSE).

To establish the context for this session, click on the best of the choices given at the left.



4531ST by Pat Lawlis

Tool Set

Compilation system

Target code tools

CASE tools

Documentation tools

Testing tools

Project mgmt tools

A tool set is a group of tools which work together to perform a particular function or a set of related functions

Choose one of the given tool sets by clicking on it.



ASSIST by Pat Lawlis

Application Area

Hard real time

Soft real time

Math intensive

Information intensive

General

The application area is a very good indicator of the features and criteria which are important when making a selection.

Choose the application area in which you are interested by clicking on it.

ASSIST by Pat Lawlis

Important Features of Compilation system

☒ contractual matters

☒ cost

☐ management functions

☐ source code sizing

☒ user profile

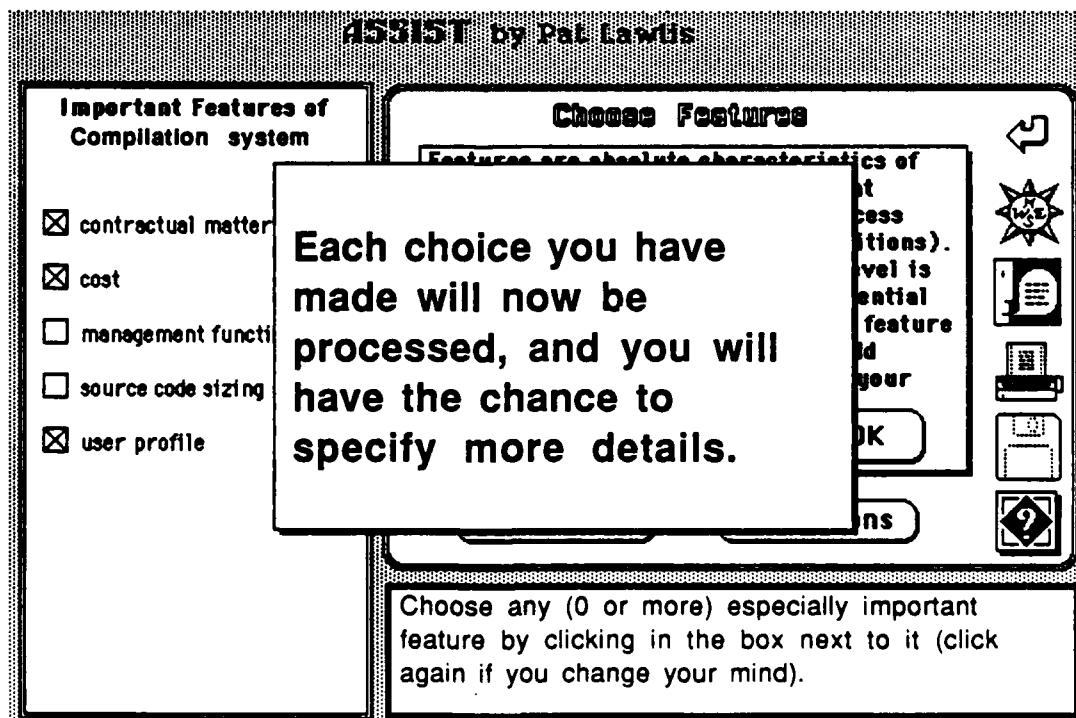
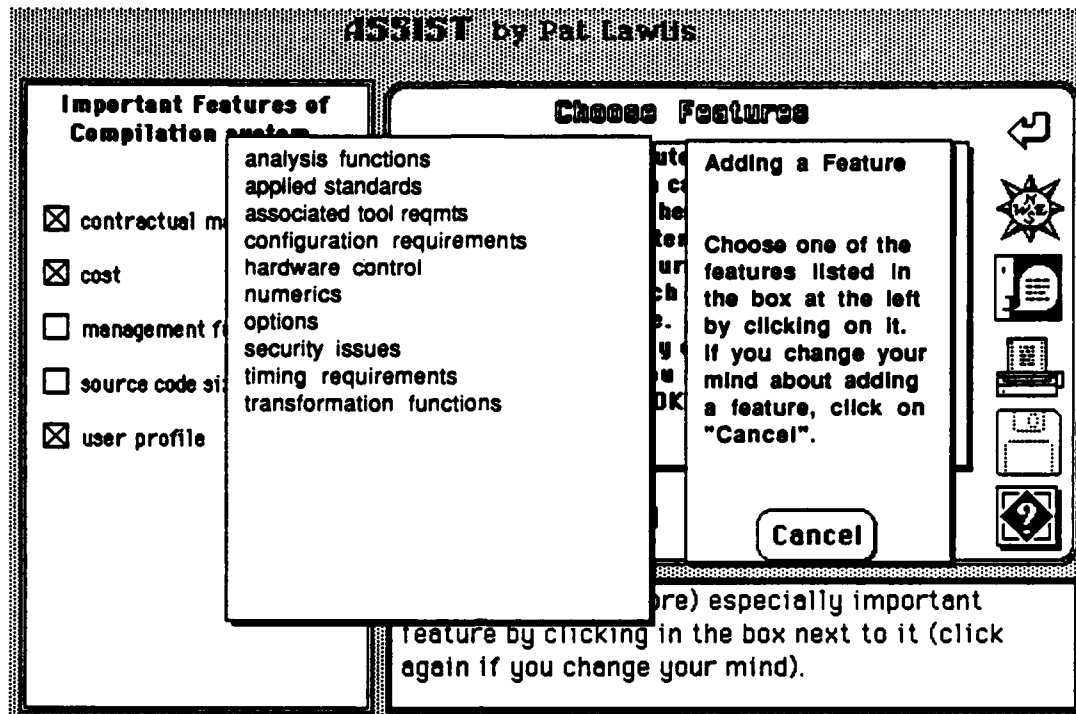
Choose Features

Features are absolute characteristics of the software which can be important considerations in the selection process (contrast with criteria - see definitions). Choose up to 8 features (this top level is very general) which are either essential or highly desirable. You may add a feature if it is not listed by clicking on "Add Feature". When you have made all your choices, click on "OK".

OK

Add Feature Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).



Management-Oriented Scenario 6

ASSIST by Pat Lawlis

Important Features of contractual matters

- ☐ number of users
- ☐ number of CPUs
- ☒ no restrictions on users
- ☐ sale of derived software
- ☐ source code available
- ☒ support available

Choose Feature Details

Features are absolute characteristics of the software which can be important considerations in the selection process (contrast with criteria - see definitions). Choose any number of the given features which are either essential or highly desirable. When you have made all your choices, click on "OK".

OK

Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).








ASSIST by Pat Lawlis

Important features of contractual matters

- ☒ skill
- ☒ training

For skill level choose from the following:

expert

intermediate

novice

choices will be used by the system. To start over from the original set of choices, unchoose all features and click on "OK", then click on "Look again".

OK

Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).








Management-Oriented Scenario 7

ASSIST by Pat Lawlis

Important Features of user profile ☒ skill level intermediate ☒ training moderate	Choose Feature Details If you unchoose features which have been chosen and/or change feature values, be sure to click on "OK" again so your new choices will be used by the system. To start over from the original set of choices, unchoose all features and click on "OK", then click on "Look again". OK Definitions Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

ASSIST by Pat Lawlis

Important Features of Compilation system ☐ contractual matters ☒ cost <= 10000 U.S. Dollars ☐ management functions ☐ source code sizing ☐ user profile	Choose Features Review the details you have specified, clicking on the magnifying glass where appropriate. When you are satisfied with all choices, click on "Done". Done Add Feature Definitions Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

Management-Oriented Scenario 8

ASSIST by Pat Lawlis

Weights of Features for Compilation system

contractual matters	5
cost	10
<= 10000 U.S. Dollar user profile	5

Weight Features

Suggested weights are given for each of the features chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK".

1

5

10

Marginally
important

Moderately
important

Critically
important
(essential)

OK

Definitions

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

ASSIST by Pat Lawlis

Weights of Features for Compilation system

contractual matters	
cost	
<= 10000 U.S. Dollar user profile	

Weight Features

The weights for each set of detail features will now be considered.

1

5

10

Marginally
important

Moderately
important

Critically
important
(essential)

OK

Definitions

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

Management-Oriented Scenario 9

45315T by Pat Lawlis

Weights of Features for user profile		Weight Feature Details	
skill level	5	Suggested weights are given for each of the features chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK". 1 5 10 Marginally important Moderately important Critically important (essential) OK Definitions For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.	
intermediate	5		
training moderate			

Weights of Features for Compilation system		Weight Features	
contractual matters	5	Review the weights for all features chosen. Click on the magnifying glass, where appropriate, for the weights of feature details. When satisfied with all weights, click on "Done". 1 5 10 Marginally important Moderately important Critically important (essential) Done Definitions For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.	
cost			
<= 10000 U.S. Dollar	10		
user profile	5		

Management-Oriented Scenario 10

ASSIST by Pat Lawlis

Important Criteria for Compilation system ☒ correctness ☐ integrity ☐ maintainability ☐ transportability ☒ usability ☒ vendor support	Choose Criteria Criteria are relative characteristics for comparing similar implementations of software (contrast with features - see definitions). Choose up to 8 general criteria which are especially important for comparing various software implementations. You may add a criterion if it is not listed by clicking on "Add Criterion". When you have made all your choices, click on "OK". OK Add Criterion Definitions Choose any (1 or more) criterion which is especially important by clicking in the box next to it (click again if you change your mind).

ASSIST by Pat Lawlis

Important Criteria for Compilation system ☒ correctness ☐ integrity ☐ maintainability ☐ transportability ☒ usability ☒ vendor support ☒ reliability	Choose Criteria Criteria are relative characteristics for comparing similar implementations of software (contrast with features - see definitions). Choose up to 8 general criteria which are especially important for comparing various software implementations. You may add a criterion if it is not listed by clicking on "Add Criterion". When you have made all your choices, click on "OK". OK Add Criterion Definitions Choose any (1 or more) criterion which is especially important by clicking in the box next to it (click again if you change your mind).

ASSIST by Pat Lawlis

Context Choices	HELP!!!		     
Application Area	Weight Criteria:		
Choose Features	Suggested weights are given for each of the chosen criteria. If you do not wish to change any of these, just click on "OK".		
Weight Features	If you wish to change the weight for a criterion, click on the weight. This will produce a dialog box which will give you the opportunity to type in the weight you wish to use. Once the weights are as you want them, click on "OK". THE WEIGHTS WILL NOT TAKE EFFECT UNTIL YOU HAVE CLICKED ON THE "OK" BUTTON.		
Choose Criteria	Click on any of the rounded buttons below for additional information about specific topics.		
Weight Criteria	Processing Details Glitches		
Display Information	For additional help topics, click on topic name at left. Click on "Icon Help" in the lower right corner for help on how the icon (picture) "buttons" work. Quit Help Quit ASSIST		Icon Help

ASSIST by Pat Lawlis

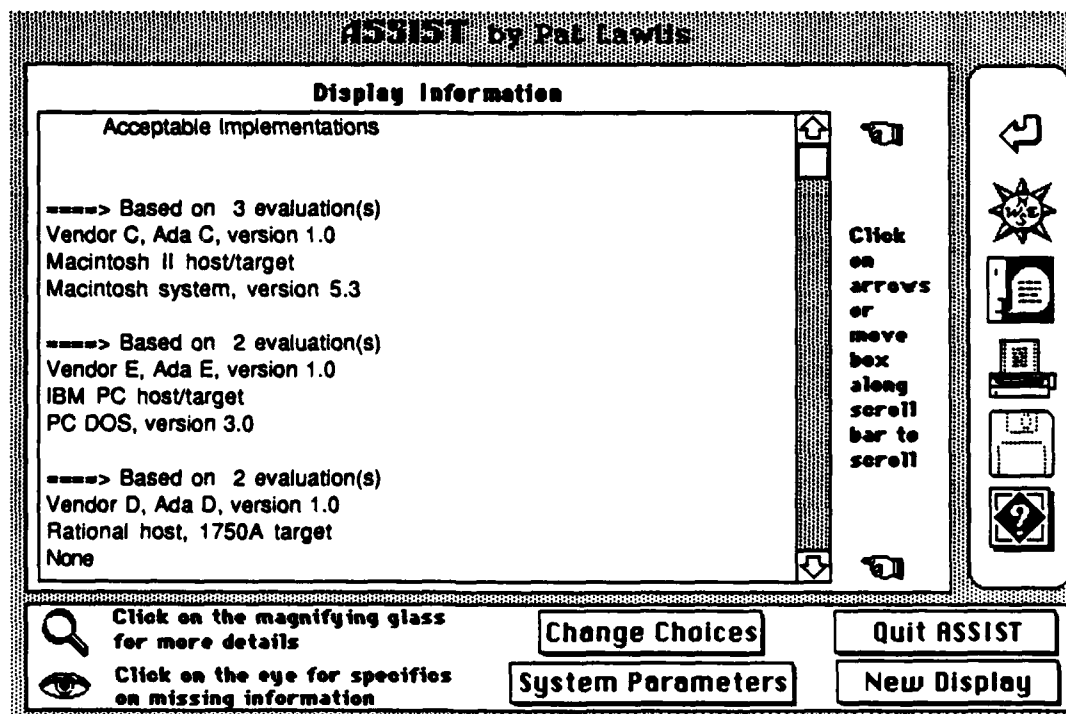
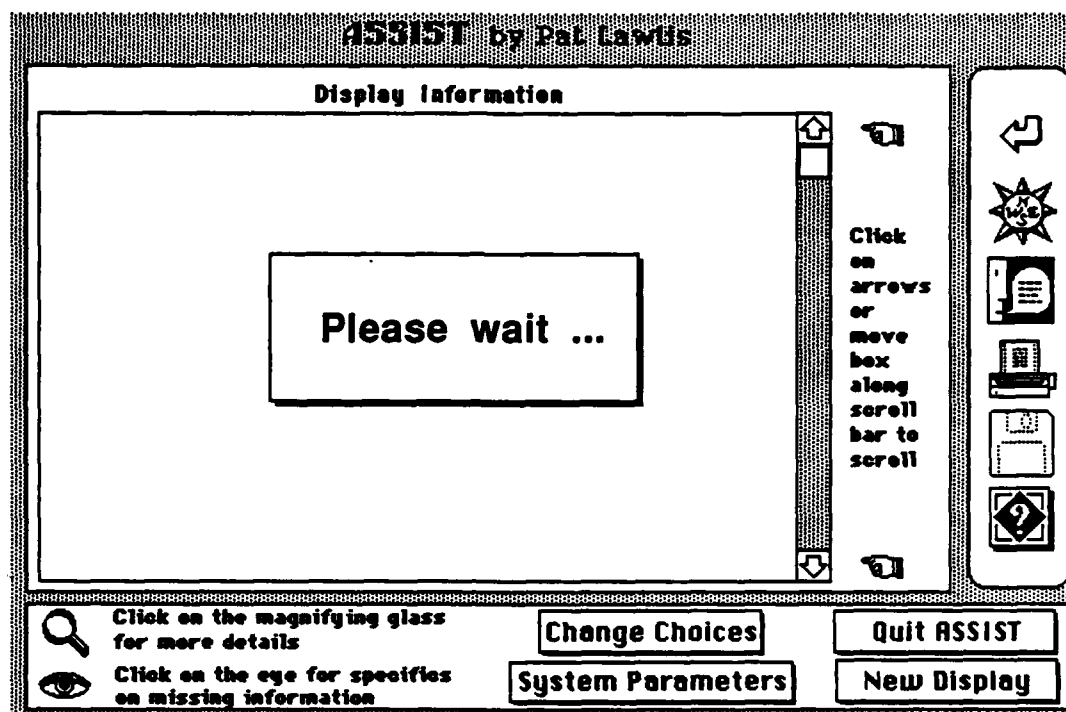
Context Choices	HELP!!!		     
Application Area	Choose Criteria:		
Choose Features	Any number of criteria may be chosen from those listed in the "Choose Criteria" window. Once all choices are made from these general criteria, a click on "OK" takes you to the windows for specifying details for each chosen criterion (if you choose to specify details). Weights will be assigned to each criterion after all details have been specified. Click on any of the rounded buttons below for additional information about specific topics.		
Weight Features			
Choose Criteria			
Weight Criteria	Making Changes Starting Over Limits		
Display Information	Processing Details No Details Glitches		Icon Help

Management-Oriented Scenario 12

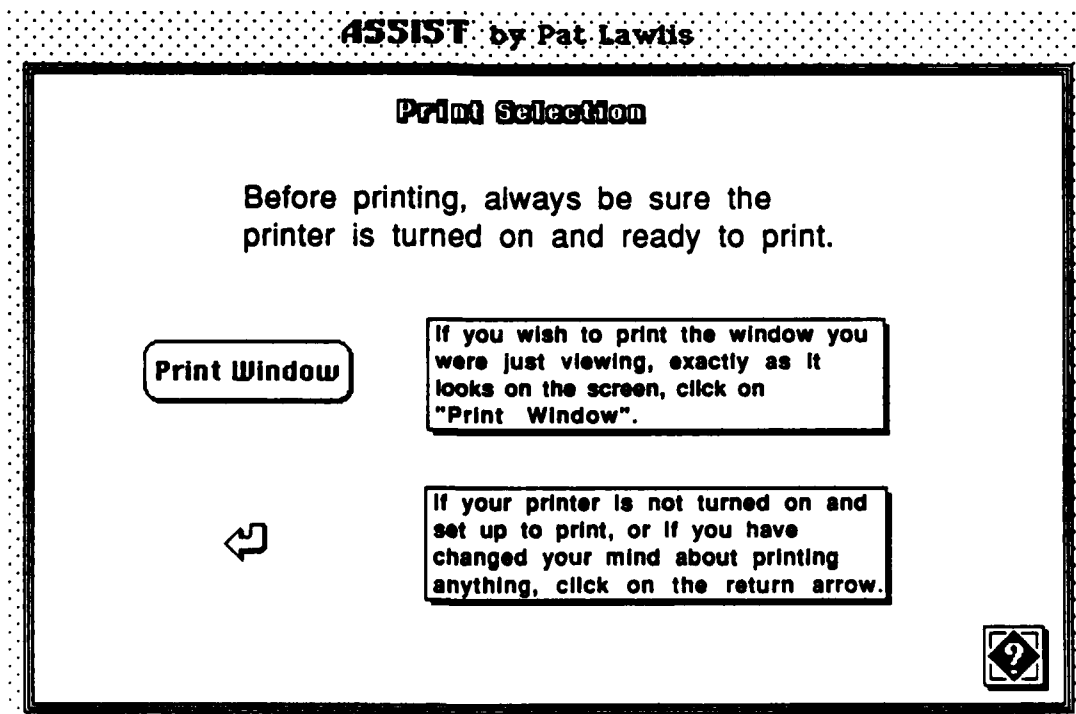
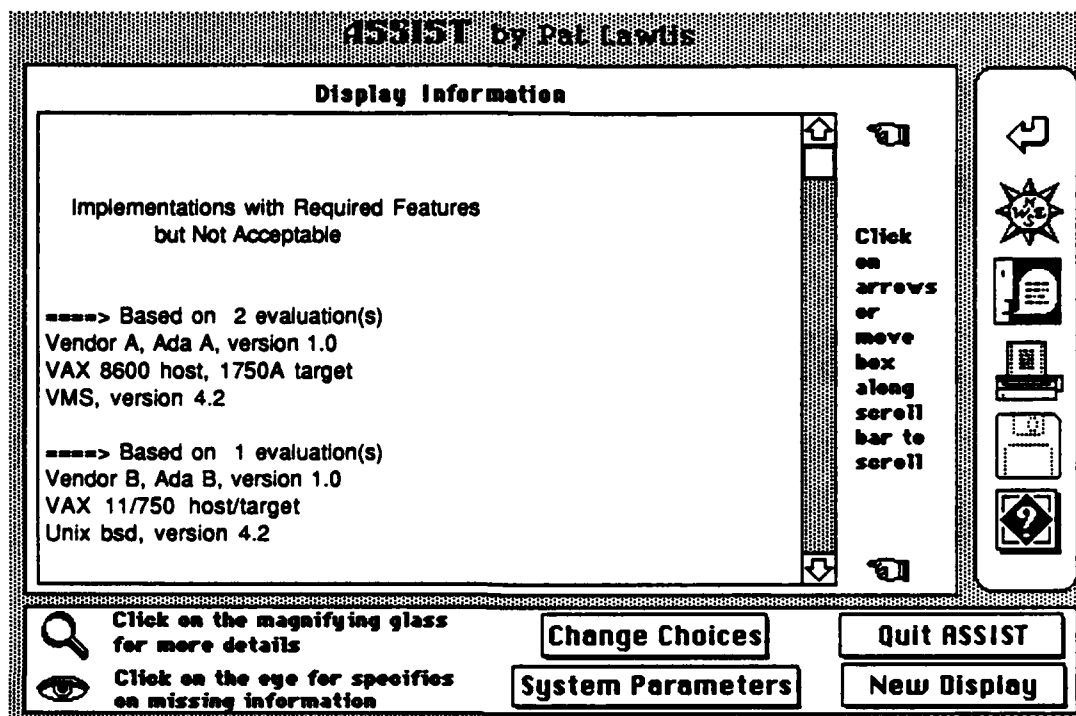
ASSIST by Pat Lawlis

Context Choices	HELP!!! No Details: You may choose not to specify criteria details by clicking on "No" when asked if you want to specify the details. In this case, all details for the general criteria you have chosen will automatically be used as long as the products under consideration have data for those details in the database.	
Application Area		
Choose Features		
Weight Features		
Choose Criteria		
Weight Criteria		
Display Information	For additional help topics, click on topic name at left. Click on "Icon Help" in the lower right corner for help on how the icon (picture) "buttons" work. Quit Help Quit ASSIST	Icon Help

ASSIST by Pat Lawlis																		
Weights of Criteria for Compilation system <table> <tr> <td>correctness</td> <td align="right">5</td> </tr> <tr> <td>usability</td> <td align="right">8</td> </tr> <tr> <td>vendor support</td> <td align="right">8</td> </tr> <tr> <td>reliability</td> <td align="right">5</td> </tr> </table>	correctness	5	usability	8	vendor support	8	reliability	5	Weight Criteria Suggested weights are given for each of the criteria chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK". <table> <tr> <td align="center">1</td> <td align="center">5</td> <td align="center">10</td> </tr> <tr> <td align="center"> </td> <td align="center"> </td> <td align="center"> </td> </tr> <tr> <td align="center">Marginally important</td> <td align="center">Moderately important</td> <td align="center">Critically important</td> </tr> </table> OK Definitions For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.	1	5	10				Marginally important	Moderately important	Critically important
correctness	5																	
usability	8																	
vendor support	8																	
reliability	5																	
1	5	10																
Marginally important	Moderately important	Critically important																



Management-Oriented Scenario 14



Appendix N

Windows Seen In Technically-Oriented Scenario

Technically-Oriented Scenario 1

ASSIST by Pat Lawlis

Life Cycle Activity

Concept development

Requirements definition

Design

Implementation

Testing

Maintenance

Life cycle activities are those activities which relate to a particular subset of the life cycle (sometimes referred to as a life cycle "phase").

Choose the life cycle activity on which you wish to concentrate by clicking on it.

ASSIST by Pat Lawlis

Important Features of Compilation system

☐ analysis functions

☐ cost

☒ source code sizing

☒ timing requirements

☐ user profile

☒ configuration requirements

Choose Features

Features are absolute characteristics of the software which can be important considerations in the selection process (contrast with criteria - see definitions). Choose up to 8 features (this top level is very general) which are either essential or highly desirable. You may add a feature if it is not listed by clicking on "Add Feature". When you have made all your choices, click on "OK".

OK

Add Feature Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

Technically-Oriented Scenario 2

45315T by Pat Lawlis

Important Features of source code sizing ☒ lines in unit ☒ units in compile ☒ entries in task ☐ elements in aggregate ☐ discriminants in record ☐ alternatives in case ☐ alternatives in select ☒ instantiations of generic	Choose Feature Details Features are absolute characteristics of the software which can be important considerations in the selection process (contrast with criteria - see definitions). Choose any number of the given features which are either essential or highly desirable. When you have made all your choices, click on "OK". OK Definitions Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

45315T by Pat Lawlis

Important Features of Compilation system ☐ analysis functions ☐ cost ☒ source code sizing ☒ timing requirements ☐ user profile ☒ configuration requirements	Choose Features Review the details you have specified, clicking on the magnifying glass where appropriate. When you are satisfied with all choices, click on "Done". Add Feature Definitions Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

Technically-Oriented Scenario 3

ASSIST by Pat Lawlis

Important Features of source code sizing

- ☒ lines in unit
 >= 5000
- ☒ units in compile
 >= 500
- ☒ entries in task
 >= 20
- ☒ instantiations of generic
 >= 10

Choose Feature Details

If you unchoose features which have been chosen and/or change feature values, be sure to click on "OK" again so your new choices will be used by the system. To start over from the original set of choices, unchoose all features and click on "OK", then click on "Look again".

OK
Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

ASSIST by Pat Lawlis

Weights of Features for Compilation system

source code sizing	5
timing requirements	7
configuration requirements	5

Weight Features

Suggested weights are given for each of the features chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK".

1
5
10

Marginally important
Moderately important
Critically important (essential)

OK
Definitions

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

Technically-Oriented Scenario 4

AS315T by Pat Lawlis

Weights of Features for source code sizing

lines in unit	5
>= 5000	
units in compile	10
>= 500	
entries in task	5
>= 20	
instantiations of generic	5
>= 10	

Weight Feature Details

You may make changes to the weights at any time, but be sure to click on "OK" again if you want the new weights to be used by the system.

1
5
10

Marginally important
Moderately important
Critically important (essential)

OK

Definitions

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

AS315T by Pat Lawlis

Weights of Features for Compilation system

source code sizing	5
timing requirements	7
configuration requirements	5

Weight Features

Review the weights for all features chosen. Click on the magnifying glass, where appropriate, for the weights of feature details. When satisfied with all weights, click on "Done".

1
5
10

Marginally important
Moderately important
Critically important (essential)

Done

Definitions

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

Technically-Oriented Scenario 5

4531ST by Pat Lawlis

Important Criteria for Compilation system

- ☒ correctness
- ☒ efficiency
- ☒ integrity
- ☐ maintainability
- ☒ reliability
- ☐ usability
- ☐ vendor support
- ☐ verifiability

Choose Criteria

Criteria are relative characteristics for comparing similar implementations of software (contrast with features - see definitions). Choose up to 8 general criteria which are especially important for comparing various software implementations. You may add a criterion if it is not listed by clicking on "Add Criterion". When you have made all your choices, click on "OK".

OK

Add Criterion
Definitions

Choose any (1 or more) criterion which is especially important by clicking in the box next to it (click again if you change your mind).

4531ST by Pat Lawlis

Important Criteria for correctness

- ☒ completeness
- ☐ consistency
- ☒ traceability

Choose Criteria Details

Criteria are relative characteristics for comparing similar implementations of software (contrast with features - see definitions). Choose any number of the given criteria which are especially important for comparing various software implementations. When you have made all your choices, click on "OK".

OK

OK
Definitions

Choose any (1 or more) criterion which is especially important by clicking in the box next to it (click again if you change your mind).

Technically-Oriented Scenario 6

ASSIST by Pat Lawlis

Definitions		
Features are characteristics of software which are used to specify absolute requirements for software implementations. Features activities transformation Ada (MIL-STD-1815A) alternatives in case alternatives in select analysis functions applied standards associated tool reqmts bits in float bits in exponent bits in integer CAIS (MIL-STD-1838A) clock resolution compiling lines of code configuration management Scroll through either the Features or Criteria (drag the box or click on the arrows) until you find the term to be defined. Then click on the term. Click on the return arrow when done.		Criteria are characteristics of software which are used to make relative comparisons of similar software implementations. Criteria accuracy application independence augmentability autonomy capacity communication commonality communication effectiveness communicativeness completeness consistency corporate health correctness data commonality distributedness

ASSIST by Pat Lawlis

Definitions		
Features are characteristics of software which are used to specify absolute requirements for software implementations. Features activities transformation Ada (MIL-STD-1815A) alternatives in case alternatives in select analysis functions applied standards associated tool reqmts bits in float bits in exponent bits in integer CAIS (MIL-STD-1838A) clock resolution compiling lines of code configuration management Integrity The extent to which unauthorized access to or modification of software or data can be controlled. OK		Criteria are characteristics of software which are used to make relative comparisons of similar software implementations. Criteria accuracy application independence augmentability autonomy capacity communication commonality communication effectiveness communicativeness completeness consistency corporate health correctness data commonality distributedness

Technically-Oriented Scenario 7

ASSIST by Pat Lawlis

Important Criteria for Compilation system

- ☒ correctness
- ☒ efficiency
- ☐ integrity
- ☐ maintainability
- ☒ reliability
- ☐ usability
- ☐ vendor support
- ☐ verifiability

Choose Criteria

Review the details you have specified (if any), clicking on the magnifying glass where appropriate. When you are satisfied with all choices, click on "Done".

Done

Choose any (1 or more) criterion which is especially important by clicking in the box next to it (click again if you change your mind).

ASSIST by Pat Lawlis

Weights of Criteria for Compilation system

correctness	5
efficiency	8
reliability	7

Weight Criteria

Suggested weights are given for each of the criteria chosen. The weights range from 1 through 10 (see below). When all weights are as desired, click on "OK".

1
5
10

Marginally important
Moderately important
Critically important

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.

Technically-Oriented Scenario 8

ASSIST by Pat Lawlis

Weights of Criteria for Compilation system

	correctness	5
	efficiency	8
	reliability	7

Weight Criteria

Review the weights for all criteria chosen. Click on the magnifying glass, where appropriate, for the weights of criteria details. When satisfied with all weights, click on "Done".

1
5
10

Marginally important
Moderately important
Critically important

Done

Definitions

For each weight you wish to change, click on the weight (the line will be highlighted), and then type in the weight you desire, as directed.








ASSIST by Pat Lawlis

Display Information

..... System Calculations

Each detail feature and criterion is rated (using the data in the database) with a 2 (good), 1 (fair), or 0 (poor).

Each detail rating is multiplied by the weight assigned to the detail feature or criterion (dwi), resulting in a weighted detail rating (wdri).

Each top level feature and criterion (TRi) is rated by summing the weighted detail ratings of its details and dividing the result by the sum of the weights of its details.

$$TRi = (wdr1 + wdr2 + \dots + wdrn) / (dw1 + dw2 + \dots + dwn)$$

Each top level feature and criterion rating is multiplied by the weight assigned to it (TWi), resulting in a weighted toplevel rating (WTRI).

The overall feature rating (OFR) is the result of summing the








Previous Display

Change Choices

Quit ASSIST



Click on the eye for specifics on missing information

System Parameters

New Display

Click on arrows or move box along scroll bar to scroll

Technically-Oriented Scenario 9

ASSIST by Pat Lawlis

Display Information

***** Current numerical ratings calculated *****

====> Based on 2 evaluation(s)
 Vendor D, Ada D, version 1.0
 Rational host, 1750A target
 None
 Rating is 1.6

====> Based on 4 evaluation(s)
 Vendor F, Ada F, version 1.0
 VAX Station host, 1750A target
 Unix bsd, version 4.3
 Rating is 1.0

====> Based on 2 evaluation(s)
 Vendor A, Ada A, version 1.0
 VAX 8600 host, 1750A target



Previous Display

Click on the eye for specifics on missing information

Change Choices

Syst. Parameters

Quit ASSIST

New Display

Click on arrows or move box along scroll bar to scroll











ASSIST by Pat Lawlis

Display Information

***** Missing Information *****

Missing data about a software product can have a significant detrimental effect on the recommendations ASSIST can make about the product.

If evaluations of critical features and criteria are missing from the database, it may be desirable to gather this data and add it to the database before making a final decision.

The following lists each of the software products considered for the current recommendations, followed by a list of chosen features and criteria for which there is no data in the database

Vendor F, Ada F, version 1.0
 VAX Station host, 1750A target



Click on the magnifying glass for more details

Previous Display

Change Choices

System Parameters

Quit ASSIST

New Display

Click on arrows or move box along scroll bar to scroll











Technically-Oriented Scenario 10

ASSIST by Pat Lawlis

Display Information

Vendor F, Ada F, version 1.0
VAX Station host, 1750A target
Unix bsd, version 4.3

Missing Features

operating system

Missing Criteria

traceability
accuracy

Click on arrows or move box along scroll bar to scroll



Click on the magnifying glass for more details

Change Choices

System Parameters

Quit ASSIST

New Display

Previous Display

ASSIST by Pat Lawlis

Important Features of Compilation system

-  source code sizing
-  timing requirements
- ☐ configuration requirements

Choose Features

Features are absolute characteristics of the software which can be important considerations in the selection process (contrast with criteria - see definitions). Choose up to 8 features (this top level is very general) which are either essential or highly desirable. You may add a feature if it is not listed by clicking on "Add Feature". When you have made all your choices, click on "OK".

OK

Add Feature

Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

Technically-Oriented Scenario 11

ASSIST by Pat Lawlis

Important Features of Compilation system

 source code sizing

 timing requirements

☐ configuration requirements

Choose Features

Review the details you have specified, clicking on the magnifying glass where appropriate. When you are satisfied with all choices, click on "Done".

Done

Add Feature

Definitions

Choose any (0 or more) especially important feature by clicking in the box next to it (click again if you change your mind).

ASSIST by Pat Lawlis

..... Type of software to be selected

Compilation system

..... Application area chosen

Soft real time

..... Features chosen and respective weights

source code sizing, 5
 lines in unit, 5
 units in compile, 10
 entries in task, 5
 instantiations of generic, 5

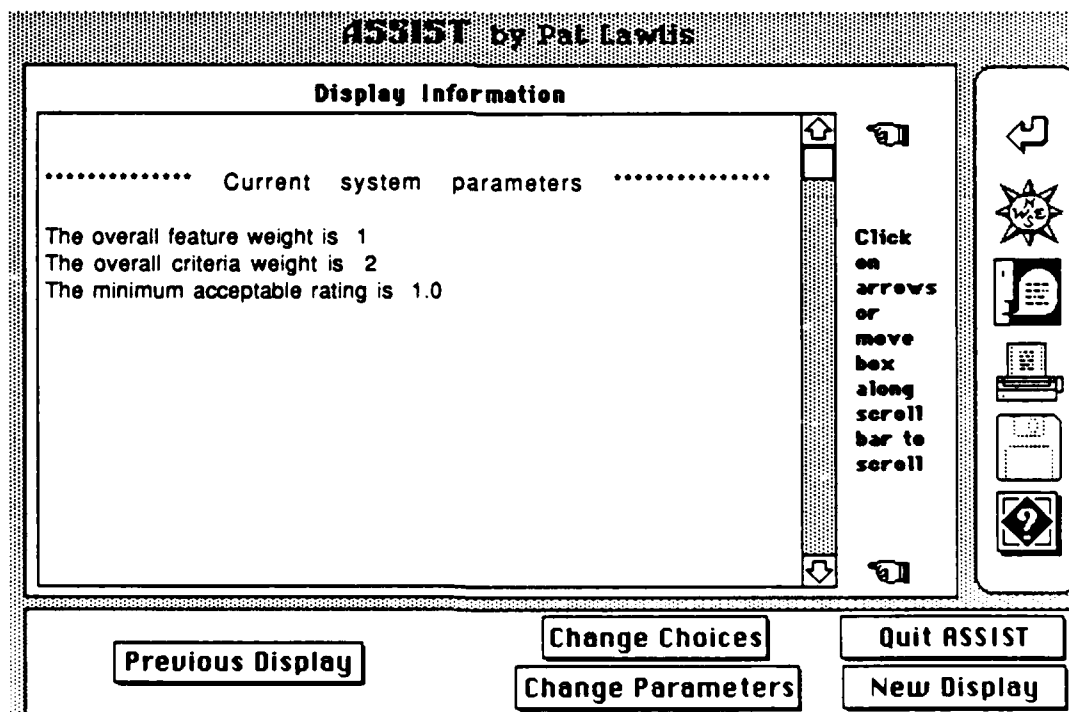
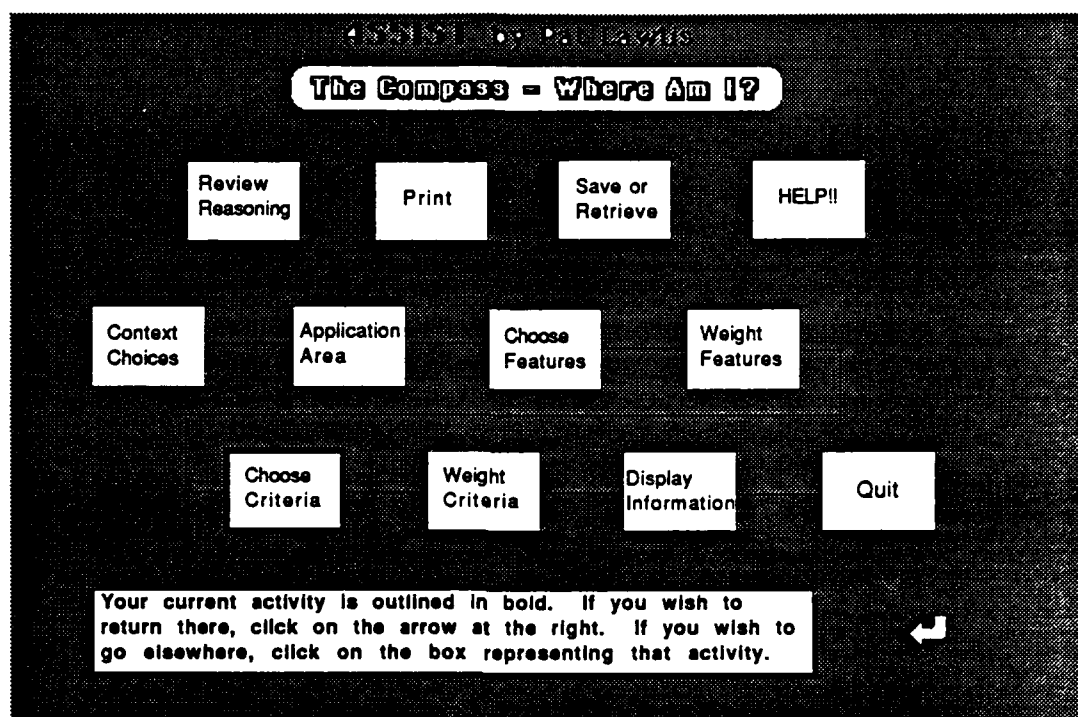
timing requirements, 7
 task rendezvous, 10
 max blocking time, 5

Review

Scroll by clicking on the arrows or moving the box along the scroll bar. When you have finished reviewing, click on the return arrow below.

Technically-Oriented Scenario 12



ASSIST by Pat Lawlis**Save or Retrieve Choices****Save**

To save the choices you have made in this session so far, click on "Save".

Retrieve

To retrieve choices which have already been saved from a previous session, click on "Retrieve".



If you change your mind, click on the return arrow to go back to the previous window.



Appendix O

Notes on the ASSIST Prototype Version 2.0

Notes on the ASSIST Prototype Version 2.0

Please comment on any or all of the following as you are working with the ASSIST prototype. Thank you for your valued feedback.

1. Does the system establish all the necessary characteristics for the software to be selected (type of software, application area, features, criteria, etc.)? If not, what should be added, deleted, or changed?
2. Is the weighting process used appropriately? If not, please suggest appropriate changes.
3. Does the system provide the right type of information to help a decision maker select software? If not, what has not been accounted for? Please be as specific as possible.
4. Do you find the system easy to use? Please comment on the user interface, the User Manual, and/or the on-line Help.
5. Please add any other comments or suggestions which you think are pertinent. Use the back if necessary.

Biographical Sketch

Patricia Kite Lawlis was [REDACTED]
[REDACTED] May 5, 1945. She graduated as valedictorian from Charleroi Area High School in Charleroi, Pennsylvania, in 1963. After spending three semesters at the Illinois Institute of Technology, she transferred to and subsequently received a Bachelor of Science degree in Mathematics from East Carolina University in 1967. After graduation, she worked as a counselor for the Pennsylvania State Employment Service in Washington, Pennsylvania for two years, and then taught mathematics at Fort Cherry High School in McDonald, Pennsylvania for five years. In July 1974 she entered the United States Air Force, and was subsequently commissioned as a Second Lieutenant. She was first assigned to Ent Air Force Base in Colorado Springs, Colorado, where she spent three years working as a computer systems design engineer with a team developing a computer system for tracking satellites. From there she went to the Programming Center Birkenfeld, a joint United States Air Force-German Air Force unit, at Birkenfeld, Germany, where she corrected the mathematics software for an air defense system. In 1980 she was assigned to the Air Force Institute of Technology at Wright-Patterson Air Force Base, Ohio, where she subsequently received a Master of Science degree in Computer Systems, as well as the Mervin E. Gross Award for outstanding scholarship, in March 1982. She stayed there and joined the faculty in the Department of Mathematics and Computer Science. In 1985 she received the Professor Ezra Kotcher Award for faculty leadership. In August 1986 she was assigned to Arizona State University for the purpose of studying for a Doctor of Philosophy degree in Computer Science. She currently holds the rank of Major. She is a member of the Association for Computing Machinery, the Computer Society of the Institute of Electrical and Electronics Engineers, Tau Beta Pi, and Upsilon Pi Epsilon.